

Jesse Liberty
Mike Kraley

ייעוץ מקצועי:
שי עדן

que



<?xml version="1.0"?>



<xsl: template match="*">

<xsl:pi name="xml">version="1.0"</xsl:pi>

XML

למפתחי אתרים באינטרנט

XML

למפתחי אתרים באינטרנט

קרא בעיון

את הוראות ההתקנה של התקליטור

בנספח ה' ובקובץ ONCD שבתקליטור

עורך ראשי: **זהר עמיהוד**

ייעוץ מקצועי: **שי עדן**

כתיבת פרק 11 והנספחים א', ב' ו-ג': **שי עדן**

תרגום: **שלומי חבה**

עריכה ועיצוב: **רמה שנקלר**

עיצוב עטיפה: **שרון רז**

לשם שטף הקריאה כתוב ספר זה בלשון זכר בלבד. ספר זה מיועד לגברים ונשים כאחד ואין בכוונתנו להפלות או לפגוע בציבור המשתמשים/ות.

- **טלפון: 09-9564716**
- **פקס: 09-9571582**
- **דואר אלקטרוני**: **info@hod-ami.co.il**
- **אתר באינטרנט**: **www.hod-ami.co.il**

XML

למפתחי אתרים באינטרנט

Jesse Liberty
Mike Kraley

ייעוץ מקצועי: שי עדן



que



XML Web Documents from Scartch

By Jesse Liberty and Mike Kraley

Editor: **Z. Amihud**

הוצאת הוד-עמי

לספרי מחשבים בע"מ

ת.ד. 6108 הרצליה 46160

טלפון: 09-9564716 פקס: 09-9571582

info@hod-ami.co.il

הודפס בישראל 2000

ISBN 965-361-265-4 מסת"ב

תוכן עניינים מקוצר

תוכן עניינים

פרק 2: מעבר מ-HTML ל-XHTML 33

פרק 3: DTD, מתן חוקיות למסמך 69

פרק 6: אחסון, הבאה לתצוגה, והצגת הסיפורים 151

פרק 7: יצירת רכיבים על ידי שימוש ב-XML ו-XSL עם DHTML ..185

פרק 10: סקירת טכנולוגיות וטכניקות...............................273

פרק 11: Schema..279

נספח ב': אתרים ברשת העוסקים בנושאי XML 329

נספח ג': טבלאות נתונים..333

נספח ד': משאבים..343

הקדמה

בפרק זה:

- הנושאים שספר זה עוסק בהם, ומה עליך לדעת
- למי מיועד הספר?
- מוסכמות הנמצאות בשימוש בספר זה

ספר זה ללימוד השפה **XML** – eXtenssible Markup Language – נוקט בגישה שאינה רגילה. במקום ללמד את פרטי השפה, הוא מתמקד בבעיה מהעולם האמיתי שברצוננו לפתור.

כנהוג בתכנון פרויקט מחשוב, אנו מתחילים בניתוח ועיצוב הפרויקט, ואחר כך מנחים אותך בתהליך העיצוב של התוכנית. תוך כדי יישום הפרויקט עצמו, נלמד אותך מיומנויות תכנות וכתיבת XML. תחילה עליך להגדיר מה ברצונך להשיג, ואז תלמד את המיומנויות הנדרשות כדי להשיג את מבוקשך.

לנוחיותך, תמצא בנספח א' "מדריך מהיר ל-XML". תוכל להתחיל בחלק זה, או לחזור אליו בהמשך כדי "ליישר קו" בנושא.

כפי שהבחנת ודאי, XML היא "שפה", ולכן נתייחס אליה בהמשך הספר בגוף נקבה. בנוסף, לתת רכיבים המשויכים לרכיבי "הורים", נקרא "בנים".

הנושאים בהם עוסק ספר זה ומה עליך לדעת

ספר זה ילמד אותך להשתמש ב-XML וב-XSL כדי לנתח, לטפל, לאחסן ולהציג מסמכים ברשת האינטרנט. במהלך הלימוד נשתמש בכלים ושפות שונות: ASP, Visual Basic, VBScript, SQL Server, DHTML, CSS, JavaScript, ADO, ActiveX ואחרים, לפי העניין. עם זאת, עיקר עיסוקנו יהיו הכלים XML ו-XSL. אם אינך בקיא בטכנולוגיות האחרות שהוזכרו – אל תיבהל! הדרישות הבסיסיות תוסברנה במהלך הלימוד.

אם מעולם לא הזדמן לך לתכנת ב-ASP או להשתמש ב- Visual Basic או ב-JavaScript, או אם אינך יודע מהו ADO, תוכל ללמוד תחילה את הנושאים מהספרים הבאים ב**הוצאת הוד-עמי**:

HTML 4 למפתחי אתרים באינטרנט

JavaScript למפתחי אתרים באינטרנט

Visual Basic 6 סדנת לימוד

Visual Basic 6 ערכת כלים

ASP 3 למפתחי אתרים באינטרנט

ASP 3 ובניית אתרים ב-**Visual InterDev** סדנת לימוד

Access 2000 VBA המדריך השלם

בסיסי נתונים טבלאיים ושפת **SQL**

אם ברצונך לעקוב אחר הדוגמאות ולנסות את הקוד, עליך לדעת תחילה את הדברים הבאים:

- כיצד להתקין ולנהל IIS (Internet Information Server), או Personal Web Server
- כיצד ליצור ספריה וירטואלית בשרת האינטרנט (Web Server) שלך
- כיצד ליצור ולקרוא דפי ASP
- כיצד ליצור ולהריץ פרויקט ב- Visual Basic
- כיצד להתקין, לנהל וליצור מסד נתונים וטבלאות בשרת SQL גירסה 7 (SQL Server 7).

אם הינך בעל ידע מתאים, אתה מוכן להמשיך.

אם הנושאים הנלווים אינם מוכרים לך במידה מספקת אך ברצונך להמשיך, תמצא שהקוד כולו מוסבר, אך ייתכן שתזדקק לעזרה כדי להריץ את הקוד במחשב שלך.

עובדות מפתח אודות XML

כשתסיים את הלימוד בספר זה, תדע:

- מהי השפה XML
- למה משמשת XML
- כל מה שדרוש לדעת על Document Object Model (מודל האובייקטים של המסמך)
- מהו XML הבנוי היטב (Well Formed)
- מהו XML חוקי וכיצד ליצור DTD
- כיצד גיליונות סגנון (Style Sheets) של XSL מאפשרים לשלוט במסמכי XML
- כיצד XML מתקשרת עם HTML
- כיצד XML מתקשרת עם מסדי נתונים

בספר זה תמצא הסבר לכל אחד מרעיונות המפתח הללו, ככל שתתקדם בלימוד. בתחילה, נערוך היכרות קצרה עם XML, נלמד מהי השפה, לשם מה נוצרה ומה הם השימושים האפשריים בה. אחר כך נציג מספר דוגמאות כיצד XML יכולה לשמש לפתרון בעיות תכנות מן העולם האמיתי, ואז נתאר בעיה אחת כזו ובחלקו העיקרי של הספר נפתור בעיה זו בשלבים.

בדרך ליישום הפרויקט, נבחן בפירוט את ה-DOM (Document Object Model) של XML, ונבחן את השווה והשונה בינו לבין DOM של HTML. כמו כן, תראה כיצד לשלב מסמכי XML עם DOM וגם עם גיליונות הסגנון של XSL, וכיצד להשיג זאת על ידי שימוש ב-VBScript וב-JavaScript. תוכל לראות כיצד מסמכי XML מתקשרים עם HTML בתוך יישום אינטרנט, וכיצד מאחסנים ושולפים נתונים משרת SQL.

למי מיועד הספר?

ספר זה מיועד לך, אם אתה מעורב באופן כלשהו **בפיתוח אתרים לרשת האינטרנט**. XML תשנה לגמרי את האופן בו אתה מפתח לאינטרנט, מאחסן מסמכים, או מטפל בצורה כלשהי בתכנים של מסמכים כאלה. הספר מיועד לבעלי רקע בנושאים שמנינו קודם. הוא אינו מיועד לחסרי רקע, או בעלי רקע חלקי ב-HTML ובניית אתרים.

מוסכמות הנמצאות בשימוש בספר זה

ספר זה מכיל מספר מוסכמות ואלמנטים שמטרתם לסייע לך במציאת מידע במהירות.



כאן תוכל למצוא מידע נוסף מעבר למה שמוצג בסעיף או הרחבה של מה שנלמד בסעיף.

סיור קצר

כאן תמצא סטייה קלה מהנושא המדובר בסעיף.

ניתוח

לאחר התדפיס יופיע ניתוח והסבר שלו.

התקליטור המצורף

בתקליטור המצורף תמצא את הדברים הבאים:

- **Microsoft SQL Server 7.0 Evaluation Software 120-Day Limit on Use**
- **קוד מקור** - קבצי קוד מקור של כל הדוגמאות שבספר.
- **קטלוג HTML** של **הוצאת הוד-עמי**.

למידע נוסף קרא את נספח ה' ואת קובץ ONCD שבתקליטור

כמה מילים על שי עדן

בספר זה שי כתב את פרק 11 ואת הנספחים א', ב', ו-ג'.

שי עדן הינו מרצה בכיר במכללת מחשבים גדולה, במסלולים המובילים: תכנות יישומים לאינטרנט, Web-Master, Web-Designer, E-Commerce וביניהם XML. במסגרת עבודתו הוא פיתח חומרי לימוד בנושאים אלה. בנוסף הוא בונה אתרים עסקיים מזה מספר שנים.

שי השתתף בכתיבת ועריכת הספר "**Flash 4 למפתחי אתרים באינטרנט**" שראה אור בהוצאת הוד-עמי.

לשאלות מקצועיות על הספר תוכלו לפנות לדואר אלקטרוני:

shaieden@inter.net.il

פרק 1
מן ההתחלה עם XML

בפרק זה:

- ייעוד מחודש של מסמכים
- מהי שפת XML ומה חשיבותה עבורי?
- XML בהֶקְשֵׁר
- ניתוח ועיצוב של פרויקט
- מסע של צעד-אחר-צעד
- הצעדים הבאים

למען האמת, XML אינה הרבה יותר מאשר **מִפְרָט** (Specification).

מפרט (Specification) הוא פרוטוקול מוסכם לאופן יצירת מסמכים מסוגים מסוימים.

המפרט עשוי לפתות מחברים של מדריכים לשימוש ב-XML לצלול הישר לתוך המבנה, ולהתמקד במודל האובייקטים ובתחביר ההתקשרות עם מסמכי XML. הדבר מאוד דומה ללמידת הטיה של פעלים בצרפתית, לפני שיש לך איזו שהיא סיבה בכלל לרצות לדבר בשפה זו!

שפות מוכוונות אובייקטים (Object-Oriented Languages) הן המעטפת של המושג **מודל האובייקטים** (Object Model). מודל האובייקטים מתאר אובייקטים בשפה וכיצד הם מתייחסים ומגיבים זה לזה. במודל אובייקטים, מתוארים האובייקטים הקיימים בשפה, אופן ההירכיה בינם לבין אובייקטים אחרים, תכונותיהם של האובייקטים ושיטותיהם.

DOM - מודל האובייקטים של המסמך (Document Object Model) מתאר את האובייקטים במסמך, וכיצד הם מתקשרים זה לזה. אחד המודלים הידועים ברשת האינטרנט הוא מודל האובייקטים של מסמכי HTML, נעשה בו שימוש רחב בכתיבת DHTML ו-JavaScript. בנוסף למודל האובייקטים של HTML, קיים גם מודל אובייקטים ל-XML. ההבדל ביניהם הוא בכך שזה האחרון נקבע על ידי הכותב, ואינו מכיל אובייקטים קבועים ומוכנים מראש. במהלך הלימוד בספר נלמד בהרחבה את מודל האובייקטים של מסמכי XML.

במקום להטיח בפניך כמות נרחבת של מידע מעורפל וללא הקשר משמעותי, הבה נתחיל בתיאור בעיה עסקית מהעולם האמיתי, ואז נראה כיצד XML מציעה פתרון לבעיה זו.

בהמשך, ניישם את הפתרון וניעזר ביכולותיה של XML להקל על משימתנו.

ייעוד מחדש של מסמכים

בכל רחבי העולם מנסים עסקים רבים לפתור את אותה הבעיה: ייעוד מחדש של מסמכים (Document Re-Purposing). כיצד לוקחים מסמכים שכבר נמצאים בידינו והופכים אותם זמינים כדפי Web באינטרנט היום, ובמדיום השימושי מחר?

ייעוד מחדש של מסמכים (Document re-purposing) מתייחס ליכולת ליצור מסמכים ומידע אחר, אשר זמינים במיגוון פורמטים (Formats) על מיגוון מדיות.

לדוגמה, נדמה עצמנו לחברת אופנה. יש לנו קטלוג מוצלח של דגמי העונה שלנו השמור בקובץ Word, הכולל תמונות בהן השקענו כסף רב ותכנים מוצלחים לטעמנו. אנחנו מעוניינים להציג את קטלוג המוצרים שלנו על רשת האינטרנט, באתר החברה, עם אופציות נוספות של חיפוש, אפשרויות לקנייה דרך האתר ועוד. לשם כך, יש ליצור מסמך HTML מתאים. בנוסף, את אותו קטלוג מוצרים המוצג ברשת האינטרנט ומספק את צרכינו מאוד, אנו רוצים לפרסם על גבי תקליטור באופן שונה ויזואלית, אבל דומה בתוכנו. לאחר מכן, נרצה ליצור מקטלוג זה, קטלוג מודפס בצבע, פוסטרים, חוברות, לוח שנה וכו׳. כל מדיה כזו נדרשת לכתיבה ועיצוב בשפה שונה. כל מסמך כזה מעוצב בשיטה שונה, אבל התוכן זהה. השינויים העיקריים הם אופן השימוש במידע ועיצובו.

המטרה היא לאפשר למסמך להיראות יפה והולם לכל מדיום, ולעצב ולייעד אותו לכל קהל יעד.

הבה נבחן את הפתרונות עבור ייעוד מחדש של מסמך Word עבור רשת האינטרנט:

הדרך הקשה לייעוד מחדש של החומר לרשת האינטרנט היא להקליד מחדש כל מסמך, ועל ידי כך להעתיק אותו מהצורה בה הוא מופיע כיום למבנה של מסמך HTML. אכן, זהו הפתרון הישיר ביותר אך הוא יקר, צורך זמן ומועד לשגיאות.

חלופה אחרת, היא להשתמש ביכולותיו של מעבד התמלילים כדי לשמור את המסמך בתצורת HTML. פתרון זה מציב בעיות משלו: באמצעות Word לא ניתן ליצור דפי אינטרנט בעיצוב מעניין ומושך, ולכן דרושה עבודה רבה נוספת לעיצוב מחדש ולתיקונים.

ובאותה רמת חשיבות, לא בנייה מחדש ולא שמירה בתצורת HTML פותרים באמת את הבעיה: בפעם הבאה שנרצה להשתמש במסמך בדרך אחרת, כמו למשל להציגו במחשב כף יד (PIM), להדפיס או לשלוח בדואר אלקטרוני, נתמודד שוב עם אותה הבעיה. אפילו אם ברצוננו לשנות רק את מראה המסמך, נאלץ להתמודד עם עבודת עריכה רבה.

הדפסה לעומת עבודה מקוונת

מעבר לשיקול המעשי של אחזקת מיגוון תצוגות של אותו מסמך, אחזקת מסמכי אינטרנט בקבצי מעבד תמלילים אינה הגיונית כלל. מעבדי תמלילים נועדו ליצירת מסמכים מודפסים. הם מצטיינים בכלים להגדרת שוליים, כותרות עליונות ותחתיות, ועימוד.

עולם האינטרנט הוא עולם חדש בו קיימת דרישה גבוהה לעיצוב מושך, ולשם כך אנו נדרשים לכלים וטכניקות עיצוב חדשות, אשר מעבדי התמלילים לא יכולים לספק לנו לעת עתה.

כאשר תמונות נעות הומצאו לראשונה, קבעו במאי הקולנוע הראשונים את המצלמה על חצובה, כיוונו אותה לעבר הבימה, והסריטו סרטים של מחזות. זה היה המדיום שהם הכירו, וכך הם יישמו את הטכנולוגיה החדשה ליצירת סרטים. ד' וו' גריפית' (D. W. Griffith) היה הבמאי הראשון שהסיר את המצלמה מעל החצובה ויצר את מה שאנו מגדירים כיום כסרטים מודרניים. הצגת דפים באינטרנט דורשת שנסיר את המסמכים מעל החצובה ונשנה את דרך העבודה שלנו ואת אופן החשיבה שלנו להצגת מידע.

בעיה אחת עם לחיצה על **שמור כ-HTML** (Save As HTML) בעורך הטקסט המועדף עליך היא שאתה מקבל תצוגת אינטרנט של מסמך מודפס, אשר בדרך כלל הוא נראה עלוב וללא חיוניות.

מעבר לכך, מסמכי אינטרנט כוללים אלמנטים אקטיביים כגון היפר-קישורים, לחצנים, רשימות גלילה, מסגרות, טפסים ועוד אלמנטים בלתי שמישים בדפוס. מסמכי אינטרנט גם מעוצבים לצורך הצגה על המסך, דבר המחייב שימוש שונה למדי בתכנון ובגופנים, מאשר דרוש במדיום מודפס.

אפילו בתוך העולם של מסמכי אינטרנט, אנו עשויים לרצות שמסמך נתון יוצג במספר רב של דרכים, על פי קהל היעד ומדיום התצוגה שלנו. לקוחות מסוימים רוצים בתקצירים ובנקודות השיא בלבד; אחרים ידרשו את כל הפרטים. חלק מן הגולשים ירצו לגלוש בגירסה המעוצבת היטב של אתרך (בטכנולוגיות עיצוב מתקדמות כמו Flash, shockwave וכו'), ואלו המחפשים מידע ולא עיצוב, יבקרו בגירסה הפשוטה אך המהירה יותר. חלק יצפו במסמך ברזולוציה גבוהה במסך גדול, ואחרים יעשו זאת במסכים קטנים וברזולוציה נמוכה מאוד, דרך חלון קטן במכשיר כף היד שלהם או בטלפון סלולרי, דרך אתר WAP. עלינו לארגן, להציג ולשלוט בתוכן המסמך באופן שונה בכל אחד מהמקרים הללו.

ייעוד מחדש (Re-Purposing) - בצורה חכמה

עלינו למצוא פתרון טוב יותר מאשר "לזרוק" מסמכים מודפסים אל הרשת. במקום זאת, עלינו לפרק את המסמך לרכיביו האטומיים ואז לערב, להתאים ולשלוט בפיסות אלו, בהתאם למקום היעד להצגתם.

כדי לעשות זאת, עלינו להסיר את התלות בין תצוגה לתוכן, ובין תוכן למבנה. בקצרה, אנו זקוקים להתייחס ליחידות האטומיות של המסמך כרכיבים עצמאיים.

לבסוף, אך במידת חשיבות זהה, עלינו להיות מסוגלים לאחסן את המסמך בצורה שתאפשר להרכיבו מחדש בדרכים שונות, ככל הנדרש. עלינו להפוך את פיסות הרכיבים של המסמך לקבועות, והמשמעות: הן חייבות להימצא במסד נתונים.

מהו הגודל המתאים ליחידות האטומיות שנאחסן במסד הנתונים? בוודאי שנוכל לשמור כל תו נפרד, אך זה מגוחך ביחס לערך הנוכחי.

תווים, מילים, משפטים ואפילו פסקאות הם חסרי ערך לכשעצמם. מאמרים – קבוצות קטנות של פסקאות המעבירות מסר עקבי – הם כמעט בוודאות היחידה האטומית הנכונה, למרות שהגודל המעשי של רכיב כזה יוחלט בהתייחס ליישום המסוים. למשל, במאמרי מערכת (הידועים בכינוי NewsLetters), סיפור בודד יהיה היחידה האטומית, אך אם אנו מפרסמים ספר לאינטרנט, סביר שפרק, או אפילו תת פרק, עשויים להיות היחידה האטומית.

דוגמאות מהעולם האמיתי

קיימות דוגמאות רבות לחברות המתעמתות עם בעיית הפריסה מחדש של המסמכים הקיימים שבידיהן. אחד מלקוחותינו ביקש לסייע לו לקחת את כל המידע ממאמרי המערכת שלו ולהפוך כל מאמר לזמין ברשת. המטרה היתה להשתמש במאמר הזהה הן באינטרנט והן בדפוס, ואז להיות מסוגלים לפרוס מחדש את המאמר במהירות לכל מדיום אחר, שעשוי להיות בו עניין. ידענו, למשל, שאותם מאמרים ישמשו שוב בדפוס להפקת סקירות שנתיות.

הלקוח שלנו נזקק גם ליכולת לחפש מאמר מסוים בהסתמך על תאריך הפרסום, המחבר או הכותרת, וגם לפי מילים בגוף הטקסט. כל אחד ממאמריו נכתב ב-Word, וכל חלקיו של מאמר מערכת מסוים נמצאו בקובץ יחיד.

משימתנו היתה לחלץ את המאמרים מהקובץ, לשמור את נתוני העל (Metadata, כגון תאריך הפרסום, המחבר וכו׳), לאחסן את הכל במסד נתונים, ואז להפיק כל מאמר במבנה מתאים על פי הדרישה.

ללקוח אחר שלנו היה מערך נרחב של מאמרים בנושא בריאות. לקוח זה מנהל אתר אינטרנט אינטראקטיבי, והמאמרים הם מידע נלווה, המסופק על פי דרישה. הוא נזקק לאפשרות לסנן ולאתר קטעים במאמרים המתאימים, בהתבסס על ההקשר בו התבקשו. יתרה מזאת, מכיון שמשתמשים שונים של המערכת יכולים לבחור בין מיגוון רמות מנוי, עליו להציג רמות מידע משתנות על פי קריטריון של המשתמש המסוים. גם במקרה זה עלינו לפרק את המאמרים ליחידות אטומיות שניתן לערב, להתאים ולהציג במיגוון תצורות.

כיצד ספר זה נוצר

הסתבר שהפתרון ללקוח הראשון שלנו היה דומה לפתרון שהצגנו ללקוח השני. למעשה, ככל שהתקדמנו, הפכנו משוכנעים שלפנינו **תבנית עיצוב** (Design Pattern) קלאסית המופיעה בספר בעל ההשפעה העתידית "Design Patterns: Elements of Reusable Object Oriented Software" (בהוצאת Addison-Wesley Professional Computing, על ידי Erich Gamma, 1995), ISBN: 0201633612.

כך הגענו להכרה ששני הפתרונות לשני הלקוחות השונים לא היו רק דומים זה לזה, אלא למעשה היו התבנית לפתרון של מאות, אם לא אלפי בעיות דומות, בהן נתקלים מפתחי אתרים באינטרנט מדי יום. בשלב זה החלטנו לכתוב את הספר הנוכחי, שמטרתו ללמד XML ו-XSL בהקשר של תבנית עיצוב זו.

תבנית העיצוב

תבניות עיצוב – design patterns – מוצגות בשמן, מתוארות ואז מודגמות. השם שניתן לתבנית העיצוב שלנו יהיה Canon, שכן האספקט המרכזי של תבנית זו הוא ליצור canoncial form.

canonical form היא הצורה המוסכמת על כולם כ"תקן", והיא מתארת את מבנה הנתונים באופן קנוני, מדורג, בדומה לאופן בו מתוארות תיקיות ותת תיקיות בסייר של "חלונות".

מטה-מידע (Meta-data) הוא מידע אודות אוסף הנתונים, כמו הכותרת, המחבר, תאריך היצירה וכו'.

אנו זקוקים ל-canonical form: מבנה ושפה לאחסון המסמכים שלנו ולאחזקת התוכן וגם מטה-מידע.

השפה XML טובה במיוחד לשימוש במקרה זה, מכיון שהיא מספקת רמזים סמנטיים, בצורת **תגיות** (Tags), אודות משמעות המידע. למשל, על ידי אחסון רצף התווים <title>Earthquake Rocks Acton</title>, אנו יודעים ששלוש מילים אלו משמשות כותרת המסמך.

נאחסן את פקודות XML שלנו במסד נתונים SQL Server 7, מכיון ששרת SQL הוא הכלי הטוב ביותר לאחסון ושליפה מהירים של מסמכי אינטרנט, שנבנו מרשומות XML. הדבר יאפשר לנו לחפש, לסנן ולנהל את מסמכי XML הללו, תוך שימוש בפקודות SQL המקובלות.



אם אינך מכיר SQL ב- 100% – אל תיבהל!

פקודות SQL שדרושות ללימוד בספר זה תוסברנה במהלך הלימוד. פירוט נרחב על פקודות SQL תוכל למצוא בספר "בסיסי נתונים טבלאיים ושפת SQL" בהוצאת הוד-עמי.

אם כך, הרי לפניך ה-canonical pattern: עליך להמיר את המסמכים ל-XML, לערוך אותם ליחידות מידע שניתן לאחסן במסד נתונים של SQL, ואז להפיק את יחידות המידע האלו ממבנה XML לתצורה הנדרשת, כמו למשל HTML, PDF, WAP וכו'.

בספר זה תלמד כיצד לעשות זאת. כפי שניתן לראות, התבנית מתמקדת ב/-XML, אחת הטכנולוגיות החדשות ואולי אף בין החזקות ברשת האינטרנט.

מה זה XML ומה חשיבותו עבורי?

חבר שלי, אשר מפתח יישומי אינטרנט מזה כמעט עשור, אמר לי לאחרונה ש-"XML תשנה את העולם הרבה יותר מאשר HTML". זו הצהרה של ממש! אם היא אכן נכונה, פירושה ש-XML עשויה להיות אחד הפיתוחים הטכנולוגיים החשובים ביותר מזה דור.

הרשת, ולצורך זה האינטרנט, היא פיתוח שהתרחש די לאחרונה. בתצוגת מחשבים בינואר 1994 ראיתי לראשונה במבט חטוף את הרשת, בביתן המציג את הדפדפן החדש Mosaic. בשש השנים שחלפו מאז, הרשת שינתה את פני העסקים באמריקה. כיום קשה למצוא חברה ללא אתר ברשת, וכל השינוי הזה התאפשר על ידי HTML בלבד. מכאן שניתן לומר בפה מלא ש-HTML, פשוטו כמשמעו, שינתה את האופן שבו אנו משתמשים במחשבים.

כיצד זה אפשרי ש-XML תהיה עוד יותר חשובה? XML יכולה להרחיב את היכולות של הרשת ממכשיר אינטראקטיבי למשלוח מידע – למדיום של **חילוף מידע** (Information Exchange Medium). משמעות השינוי היא שעסקים יוכלו לבצע יותר ממשלוח נתונים ומידע ברשת. הם יוכלו לנהל פעילות עסקית מסחרית בהיקף רב.

XML בהֶקְשֵר

במובן מסוים, הדבר הקשה ביותר ב-XML הוא ההבנה מה היא השפה המיוחדת הזו, ולמה היא משמשת. כדי להבין זאת, עלינו לדון בהיסטוריה של הדרך שבה נוצרה.

הכל מתחיל ב-SGML (Standard Generalized Markup Language, שפת סימון סטנדרטית מוכללת). התפקיד של SGML היה לתאר בניית שפת סימון על ידי התחביר וההגדרות של האלמנטים והתכונות של השפה. SGML הינה מטה-שפה.

מטה-שפה (Meta-Language) זו שפת על, המתארת שפות אחרות.

התפקיד של SGML להגדיר את התחביר של **שפת סימון** (Markup Language).

תחביר (Syntax) הוא אוסף התווים, הפיסוק והסדר של המילים בשפה.

שפת סימון מגדירה כיצד להוסיף משמעות למידע, כאשר בדרך כלל הכוונה היא למסמך. ה"סימון", מתאר את סימונו של קטע טקסט ומתן אפיון לקטע. שים לב, נהוג לבלבל בין המונח הלועזי Mark-Up, המתאר סימון של טקסט על ידי עט סימון (מארקר) למונח MarkUp, המתאר סימון על ידי שפת סימון.

אחת השפות שמתוארות בדרך זו על ידי SGML היא השפה HTML. למעשה, HTML היא תת שפה (Instance) של שפת SGML. המשמעות ש-HTML מוסיפה למסמך היא בעיקר מידע אודות **הצגה** (Presentation) של הנתונים שבו. למשל: Jesse Liberty. התגיות ו- מסמנות את הנתון "Jesse Liberty" כטקסט אשר יוצג בדפדפן באופן מודגש (Bold).

ניתן ליצור מספר כלשהו של שפות עם SGML, למרות ש-HTML היא שפת הסימון מבוססת-SGML הפופולרית ביותר שנוצרה.

שים לב ש-SGML אינה מגדירה את **המבנה** של HTML. משימה זו מושגת על ידי DTD (Document Type Definition, הגדרת טיפוס מסמך). הגדרת טיפוס מסמך HTML (כלומר, DTD) מבוססת על תחביר המוכתב על ידי SGML והתוצאה – הגדרת מסמך HTML תקף.

הגדרת טיפוס מסמך – DTD (Document Type Definition) היא התיאור של התחביר החוקי של מסמך. DTD מגדיר מהו התחביר המותר, אופן הכתיבה שלו, רמת ההיררכיה שבין אלמנטים במסמך, התכונות והערכים אשר יכולים לקבל אלמנטים במסמך. חריגה מההגדרות של המסמך, כפי שהן נקבעות בתוך ה-DTD, אינה חוקית, מסמך הנכתב על פי ההגדרות של DTD נקרא "תקף". DTD עבור HTML מגדיר את התקפות של
, <p> ושאר התגיות המשמשות ב-HTML; SGML מספקת את התחביר עבור השפה שהיא מגדירה.

בסופו של דבר, לא SGML ולא DTD מגדירים את הסמנטיקה של HTML. הסמנטיקה היא המשמעות של כל תגית, כמו למשל העובדה, שמשמעות התגית <b> היא הדגשה. משיגים זאת על ידי תוכנה מפרשת: במקרה זה, זהו הדפדפן.

סמנטיקה (Semantics) היא המשמעות של מילים וביטויים בשפה.

הבה נעשה סדר:

- SGML הינה שפת על המגדירה את התחביר של HTML.
- DTD של HTML, היא הגדרת השפה של HTML, אשר מגדירה ולמעשה גם קובעת אם מסמך נתון הוא חוקי ותקף.
- הדפדפן מפרש את המסמך ומציג אותו כפי שמורות לו התגיות.

XML מפשטת את SGML

אם כך, היכן משתלבת XML בתמונה זו? XML היא תת שפה של SGML. זהו למעשה פישוט של SGML כדי לשרת מטרה כמו זו של SGML אך היא עושה זאת ביתר קלות. הרעיון הוא לשמר כמחצית תכונות SGML שמשתמשים בהן רוב הזמן.

זו אינה אבחנה טריוויאלית. SGML היתה כה קשה לשימוש, עד שכל מי שלא היה מומחה לא יכול היה להשתמש בה ביעילות. עובדה זו מציבה סף כניסה גבוה מאוד: כדי להשתמש ב-SGML צריך להשקיע זמן רב (או כסף) בהשגת המומחיות הנדרשת. התוצאה היתה שמעט אנשים השתמשו בשפה והיא הפכה לטכנולוגיה שולית.

הנה הסוד לגבי XML: היא אינה קשה.

XML היא כה ישירה, עד שלמעשה, על ידי קריאת ספר קצר אחד בנושא (ספר זה לדוגמה.) רוב המפתחים יכולים להשתמש בה באופן שוטף ומייד. עיקר עוצמתה של XML טמון בפשטותה. מטה-שפה בעלת עוצמה יכולה לשנות את האופן שבו אנו מגיבים עם נתונים ; XML היא מטה-שפה (Meta-language) כזו.

XML מגדירה תחביר של שפות. כמו בעזרת SGML, באפשרותך להגדיר HTML על ידי XML, וניתן גם להגדיר באמצעותה עוד מספר רב של שפות חדשות אחרות. הדבר החשוב ביותר אודות XML הוא שהיא נוצרה במיוחד כדי שניתן יהיה להרחיבה. שלא כמו HTML, XML אינה מכילה אוסף קבוע של תגיות.

הדגש ב-XML אינו על תצוגה. בצורה אופיינית, תגיות XML אינן אומרות לך דבר אודות אופן הצגת הנתונים, אך הן אומרות הרבה על משמעותם. יכולת זו לספק תגיות בעלות משמעות היא שנותנת ל-XML את עוצמתה, כפי שנראה בהמשך.

כיצד נראית XML?

כשאתה רואה לראשונה מסמך XML, הוא נראה בדיוק כמו מסמך HTML, מלבד העובדה שתבחין מייד, שמעולם לא ראית את התגיות המסוימות הללו קודם לכן.

הקוד הבא הוא ציטוט חלקי מתוך קובץ XML שנעסוק בו מאוחר יותר בספר זה :

```
<book>
<section level="A" id="10000">
<title>Chapter 10</title>
<section level="B" id="10001">
<title>Leveraging the Standard Template Library</title>
```

שים לב שבמקום למצוא תגיות כמו
 ו- <div>, אתה רואה כאן תגיות מיוחדות כמו <book>, <section> ו- <title>. זוהי המהות של XML : תגיות סמנטיות בעלות משמעות.

עובדה חשובה אחת אודות תגיות XML היא, שהמחבר של מסמך XML מורשה ליצור תגיות בעלות משמעות בהקשר הפיתוח שלו עצמו. פירושו של דבר, שלתגיות זהות תהיינה משמעויות שונות במסמכים שונים. למשל, התגית <title> במסמך אחד ייתכן שתתייחס לכותרת של הפרק, אך במסמך אחר היא עשויה להתייחס לתואר (גם title באנגלית) של אדם (כמו .Dr, .Mr וכו'), בהקשר אחר, היא יכולה להתייחס לתפקידו של אדם מסוים, או לבעלות על בית או מכונית.

באופן דומה, תגית כמו <window> בהקשר מסוים, תתייחס לחלון של תוכנה, ובהקשר אחר, במסמך המפרט חלקי בית, תתייחס לחלון בבית.

יוצר המסמך הוא מי שיוצר את התגיות, והוא שנותן להן שם ומשמעות, על פי צרכיו.

תכונות של תגיות XML

ב-XML, תגיות יכולות להכיל גם תכונות (Attributes) בעלות משמעות. כשם שהתגית input של HTML יכולה להכיל את התכונות name=action ו-type=submit, כאן תגית כמו <door> יכולה להכיל את התכונות "height="300 ו-"color="red.

הבה נתעלם בינתיים מן המשמעות של התגיות המסוימות הללו ונתמקד בעובדה שהן מוגדרות כבעלות משמעות בהקשר של הפרויקט שלנו. כלומר, שפת הסימון הורחבה וכוללת עתה תגיות בעלות משמעויות המותאמות לצרכינו!

כשם שלתגית חייבת להיות משמעות בהקשר שלה, כן גם לתכונות יש לספק משמעות בהקשר. התכונות של סעיף (Section) בספר תהיינה כמובן שונות מהתכונות של אזור (גם כן Section) בתכנון עירוני.

מהי XSL?

הבנו ששפת XML הינה שפה לייצוג תוכני של מידע. אבל, ישנה חשיבות מרובה לאופן עיצוב הנתונים והצגתם בסופו של דבר למשתמש. לשם כך, יש להעביר את קוד ה-XML דרך שפת עיצוב סגנונות, אשר תשלוט באופן הצגת הנתונים למשתמש.

אחת האפשרויות היא Cascading Style Sheets) CSS) - שפת עיצוב גיליונות מדורגים, המוכרת לרובנו מ-DHTML. CSS בעלת יכולות גבוהות לעיצוב ויזואלי של נתונים, אך רמת השליטה בנתונים, לפני הצגתם נמוכה.

השפה eXtensible Stylesheet Language) XSL) - שפת גיליונות עיצוב הניתנים להרחבה - משמשת להמרה של מסמכי XML ולשליטה בתצוגה שלהם. שפה זו חשובה מאוד בעבודה עם XML, ובשונה מ-CSS, היא מכילה תכונות מתקדמות, וחזקות במיוחד לעיבוד נתונים והצגתם.

כפי שנראה בספר זה, הדפדפן Internet Explorer 5 (IE5) הינו היחיד התומך כיום ב-XML, ואפילו הוא, תומך רק במחצית מהיכולות של XSL: בהמרות (טרנספורמציות). למרבית המזל, אלו הן היכולות החשובות, והן מבטאות את ההבדל הניכר בין XSL ו-CSS. המרות על ידי XSL מאפשרות לטפל במבנה ובתוכן של מסמך XML, ובספר זה נעשה בכלי זה שימוש נרחב.

ניתוח ועיצוב של הפרויקט שלנו

לשם הלימוד, בחרנו פרויקט דמיוני, אשר העבודה עליו תלווה ביישום שפת XML ו-XSL.

בפרויקט שלנו נתחיל, כמו בכל פרויקט, בניתוח הדרישות. אחר כך נעצב פתרון, ולבסוף ניישם את הפתרון שבחרנו בו. מכיון שזו תוכנית הדגמה קטנה למדי, הניתוח יהיה פשוט יותר מאשר בעולם האמיתי. אנו להוטים להתחיל בעיצוב וביישום, אך הבה נוודא תחילה שאנו מבינים לחלוטין את הבעיה שברצוננו לפתור.

BiblioTech - דרישות הפרויקט

בתסריט הדמיוני שלנו, החליט תאגיד Que לספק את תוכן הספרים בצורה מקוונת, וליצור אתר אינטרנט שבו יוכלו קוראים לשאול שאלות במיגוון נושאים. התשובות לשאלות הקוראים יכילו ציטוט, ואפילו קטעים מתוך הספרים. כדי להשיג מטרה זו, אנו זקוקים לכלי המסוגל למצוא ולהציג במהירות, ולפי דרישה קטעי טקסט או קטעי קוד מתוך ספר.

ניתוח אירועים

אין בכוונתנו ליצור ניתוח אירועים (Use-Case Analysis) מלא ומפורט עבור פרויקט זה, מכיון שאנו מתכוונים להשאיר את הדברים מאוד פשוטים (להסבר מלא אודות ניתוח אירועים וניתוח ועיצוב מוכווני-עצמים, ראה את רשימת הקריאה המוצעת בנספח ד׳).

כעת, נגדיר את **השחקנים** (Actors) העיקריים במערכת כ״מחבר״ וה״סטודנטים״, או ה״קוראים״ המקוונים.

בניתוח ועיצוב מוכווני-עצמים, **שחקן** (Actor) הוא כל אדם, או מערכת, המגיבים ליישום שאנו מעצבים.

האירועים (Use Cases) החשובים הם:

1. המחבר משתמש ב-BiblioTech כדי למצוא הסבר הולם לשאלה שהופנתה אליו, והוא יכול לגזור ולהדביק הסבר זה הישר לתוך הודעת דואר או דף אינטרנט.
2. המחבר משתמש ב-BiblioTech כדי למצוא מובאה מן הספר בנושא מסוים, שאותה יגזור וידביק לתוך הודעה אחרת.
3. המחבר משתמש ב-BiblioTech כדי לסקור, ואולי בסופו של דבר אף לערוך קטעים אחדים הקשורים לאותו נושא.
4. הסטודנט משתמש ב-BiblioTech באמצעות רשת האינטרנט, כדי למצוא הסברים אודות נושאים מסוימים המעניינים אותו.

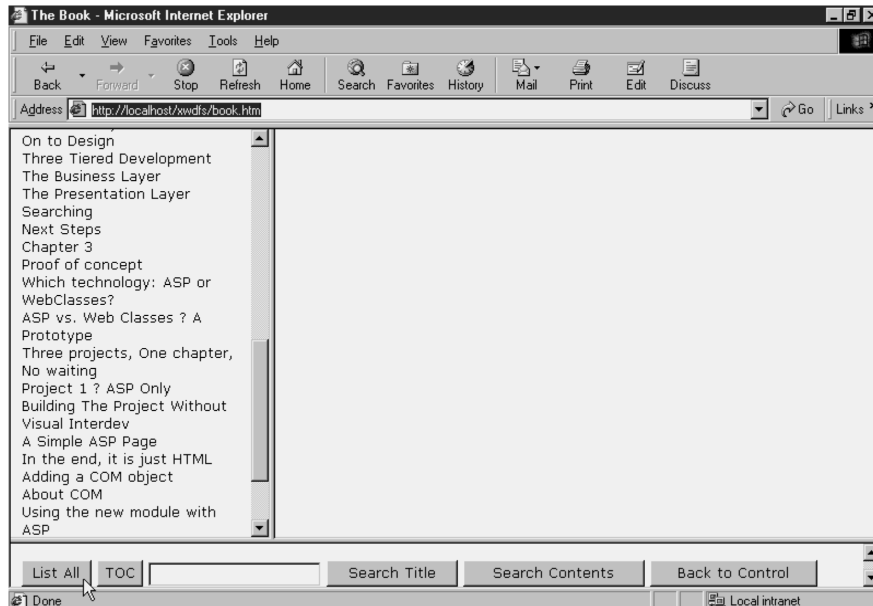
ישנם אירועים נוספים אשר ניתן לחשוב עליהם; עם זאת, אלה יספקו לנו התחלה טובה להבנת הדרישות מהמערכת שנציג בהמשך.

מראה (Visualization)

BiblioTech הוא כלי שיספק רשימה של כל הנושאים בספר, בצורת רשימה או טבלה הניתנת להרחבה. כל נושא יהיה פעיל - לחיצה על הנושא תציג את המאמר, או את קטע הטקסט הרלוונטי. כמו כן, המשתמש יוכל לחפש מילים בכותרת, או בכל מקום בטקסט.

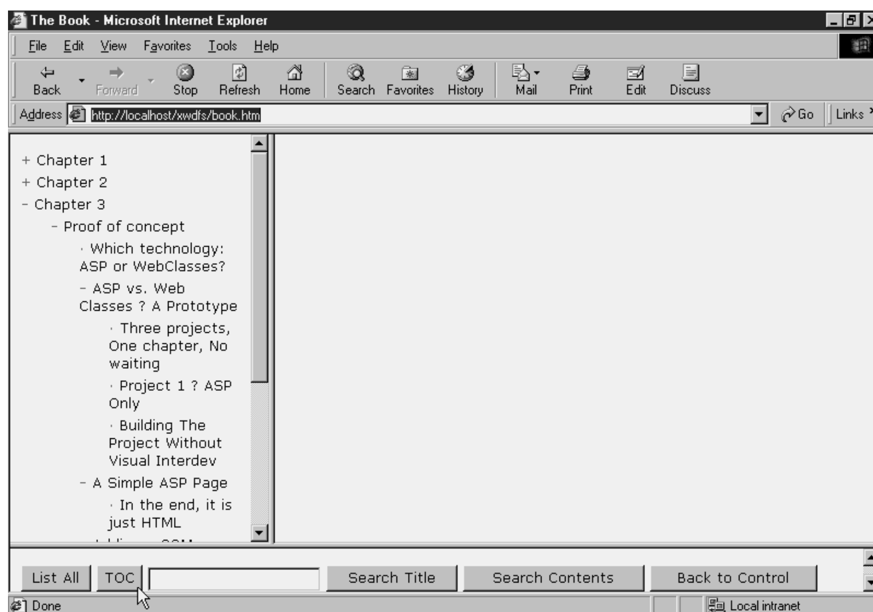
חשוב לקבל מושג על המראה של יישום זה. בכך ניתן לקבל תובנה רבת עוצמה על אופן הפעולה של היישום והשימוש בו.

להלן תרשים 1.1 הממחיש את BiblioTech כפי שאני מדמיין אותו.



תרשים 1.1: BiblioTech

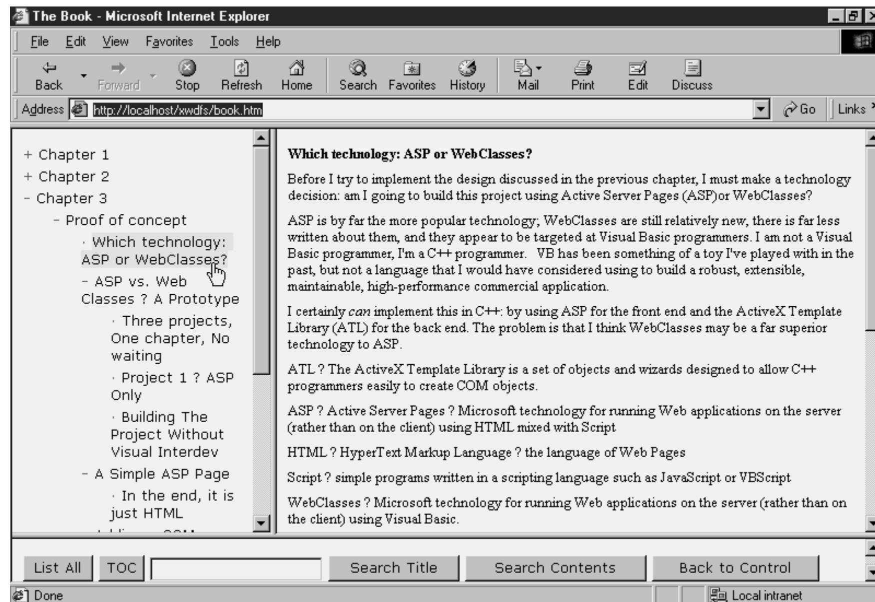
BiblioTech הוא יישום אינטרנט פשוט וברור למדי. בתחתית המסך מציבים פקדים המאפשרים למשתמש להציג את רשימת כל הנושאים (כמוצג בתרשים 1.1), או להציגם בטבלה הניתנת להרחבה (Collapsible table of contents), כמוצג בתרשים 1.2.



תרשים 1.2: טבלת תוכן הניתנת להרחבה

בנוסף לזאת, המשתמש יכול להקליד מילה או ביטוי לתוך שדה הטקסט ולחפש בכותרות. הרחבה קלה של תוכנית זו, המושארת כתרגיל לקורא, היא לאפשר חיפוש טקסט מלא, וכן גם סינון ומיון של התוכן.

אם המשתמש לוחץ על כותרת מאמר או הטקסט, הוא יוצג בחלק המסך הימני.



תרשים 1.3: תצוגת מאמר

היישום BiblioTech מסתמך במשתמע על כך שכל חלק של הספר (המוגדר כאן כמאמר, Article), מאוחסן במאגר הנתונים וזמין במבנה XML. נחזור ליישום BiblioTech לקראת סוף הספר, לכשנגיע את המטלות הללו.

מסע של צעד-אחר-צעד

מטרתנו היא לעבור למצב בו כל חלקי הספר מאוחסנים בבסיס נתונים, כך שיהיו זמינים במבנה XML. לשם כך, עלינו להמיר את קבצי ה-Word, אל מבנה הנתונים. הצעדים הבאים נדרשים כדי לעבור מקבצי Word, שכל אחד מהם מכיל פרק, למבנה הנתונים הנדרש עבור BiblioTech:

1. שמור את מסמכי Word הקיימים כ-HTML.
2. המר את קבצי HTML למבנה HTML המעוצב היטב (ידוע גם כ-XHTML, או HTML מורחב). תוכל להשיג זאת על ידי פעולות על מודל אובייקט המסמך (DOM) של HTML (HTML Document Object Model).

3. המר את קובץ XHTML למבנה XML. עשה זאת בשני שלבים: תחילה, על ידי שימוש בגיליון עיצוב של XSL ואחר כך, על ידי הסבה ישירה של מודל האובייקטים של המסמך (DOM) של XML.
4. פצל את הטקסט לנושאים או סעיפים, ושמור אותם במסד נתונים של SQL Server.
5. בנה את תצוגת טבלת תוכן העניינים.
6. בנה את היישום BiblioTech.

לאחר ביצוע הפעולות שפורטו לעיל, תוכל לשפץ את היישום. בפרקים האחרונים נראה יחידות תכנות נוספות, שתסייענה בהבנה ובהפעלת XML ו-XSL.

מה לא אציג בפרויקט

בפרויקט כזה, כמו במרבית יישומי האינטרנט, חלק ניכר מהמאמץ האמיתי מופנה ליצירת תצוגה טובה. לא אעשה זאת כאן, מכיון שמטרתי ללמד את השימוש ב-XML, ואיני מתכוון להדגיש את נושאי התצוגה. על כן השארתי את היישום הזה דל במכוון, מבחינת התצוגה. הדגש בספר זה הוא על XML ועל XSL; אין בכוונתי להסיח את תשומת הלב על ידי טריקים בתכנות, שאינם רלוונטיים למטלה הנדונה.

ואפילו כדי לבנות את היישום הדל הזה, עליי להשתמש בכלים רבים של טכנולוגיות התכנות ברשת, ובכללן JavaScript, ASP, ADO ועוד. אסביר את הכלים המתוחכמים האלה ככל שנתקדם. עם זאת, חשוב לזכור שהדגש יהיה על XML ועל XSL, ולא על הטכנולוגיות התומכות הקשורות.



מהם יתרונות השימוש ב- ASP לעומת WebClasses?

התשובה היא משולשת: ראשית, דפי ASP הם פשוטים למדי, **ואין יתרון** לשימוש ב- WebClasses (המצטיינים ביישומים מורכבים יותר).

שנית, לעת עתה, ASP הוא סביבת תכנות שכיחה בהרבה, כך שיותר קוראים בוודאי מכירים אותה.

שלישית, דפי ASP אינם מחייבים רכישת VB, **ואני רציתי להבטיח שתוכל ליישם את כל הקוד בספר זה מבלי שתצטרך לרכוש סביבת פיתוח חדשה.**

הצעדים הבאים

לאחר סקירת הפרויקט, הבה נתחיל בבחינת מסמכי Word המכילים כמה פרקים מן הספר. מטרתנו להמיר אותם ל-XML, מטרה שנשיג על ידי שמירת מסמכים אלה כ-HTML, המרתם לקבצי HTML אשר תקפים ב-XML. לבסוף, נמיר אותם ל-canonical format שלנו ב-XML.

פרק 2
מעבר מ-HTML ל-XHTML

בפרק זה:

- ניתוח
- הסבה ל-HTML
- הסבת Word ל-XHTML
- סקירת הקוד
- הצעדים הבאים

בפרק 1 עסקנו בצעדים הנדרשים להמרת מסמכי Word ל-XML. מטרתנו להגיע למסמך XML המפורק לקטעים (״סיפורים״, מאמרים, או יחידות אטומיות) ומאוחסן במסד נתונים. הדבר יספק לנו ארכיטקטורה גמישה, שממנה נוכל לשחזר ולפרוס מחדש את קטעי הטקסט לכל מבנה ומטרה שנחפוץ.

כדי להשיג זאת, עלינו לבצע סדרת המרות, או טרנספורמציות. הראשונה היא המרה ממבנה Word הבינארי המיוחד, למשהו שנוכל לעבוד אתו. עם זאת, בתהליך זה עלינו לפרש גם את משמעות המסמך, כך שנוכל להמיר מהמבנה הפנימי הקיים למבנה XML ללא אובדן של נתוני-העל (Metadata) המצויים כרגע במסמך Word. כדי להבין סוגייה זו, עיין בתדפיס 2.1 המציג קטע בו נשתמש בפרויקט זה:

תדפיס 2.1 מובאה מתוך מסמך Word מעוצב

Analysis

EmployeeNet will manage the human resources needs of middle-sized corporations. The project is being sponsored by Acme Manufacturing, and we'll build it to meet their needs and then generalize for other companies as we go.
Acme employees 2,000 people in the manufacture of high-end consumer products. Their principal products are the Acme Widget and their world-famous Gizmo. They have manufacturing plants in East Podunk and New Boondock and their main offices are in Gotham.

Conceptualization

EmployeeNet will allow the personnel department to track all benefits for employees, and will allow employees to review and edit their own employment records.

Use Cases

There was a time when the typical requirements for a software project were expressed in terms of capabilities and performance. While this ensured that the resulting system met certain specified benchmarks, it did not ensure that anyone could or would want to use it. Typically the user didn't factor into consideration until after the product was out the door.

Object-oriented analysis begins with a thorough understanding of how the product will be used. A use-case is a formal statement of the various ways in which a user will interact with the system, and what he will want to accomplish.



Use Case - A formal statement of one way in which the system will be used.

We begin the use case analysis by identifying the principal "actors" - that is the users of the system. These include

- Human resources personnel (entering and updating records)
- Human resources managers (reviewing the work done in their department and setting policies)
- Employees (reviewing their own records)
- Managers (reviewing the employee records of their direct reports)



Actor - a person or other software system who interacts with the system we're creating

---

מובאה קצרה זו חושפת מספר עובדות מעניינות ורלוונטיות אודות ספר זה. החשובה ביותר היא שמסמך Word כבר מסומן (Marked-Up) בחלקו ומותאם להפקת דפוס. כשכתבתי ספר זה, סימנתי טקסט ככותרות חלקים, כותרות פרקים, רשימות ממוספרות, הערות וכן הלאה. עשיתי זאת בשתי דרכים.

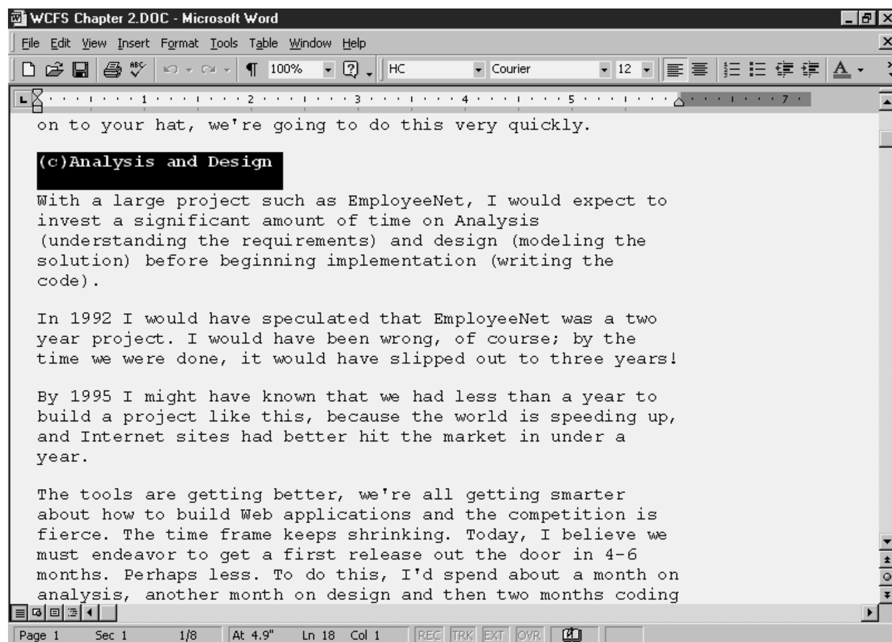
תחילה, השתמשתי בסימונים מפורשים בטקסט. למשל, בטקסט המוצג לעיל, הסעיף "ניתוח" הוא כותרת מרמה C. כותרת מרמה A היא מספר פרק, מרמה B היא כותרת פרק, וכותרת מרמה C (או Heading 1) מסמנת סעיף. כותרת מרמה D היא Heading 2, כלומר היא תת כותרת עבור כותרת מרמה C. יש גם כותרות מרמות E, F ו-G.

על ידי סימון "ניתוח" ככותרת מרמה C, ציינתי שזהו השם של סעיף מרכזי. על כן, קווי הסגנון המנחים מכתיבים שהכותרת תודפס בגופן בגודל 18 נקודות, מודגש, מסוג Helvetica Cond B באנגלית, או Arial(Hebrew) 20 בעברית, וכן הלאה. סעיף המשנה "הרעיון" הוא ברמה D, ויודפס בגופן 14 מודגש מסוג Helvetica Cond B באנגלית, או גופן Arial(Hebrew) 16 בעברית.

בנוסף לסימון הכותרות, יש להתאים את סגנון Word לכל אחת מהן. הסגנון הוא מתוך קבוצה של סגנונות המסופקת על ידי Macmillan, ומצויה בין תבניות מסמכי Word שלי.

סגנונות אלה הם בעלי משמעות רק למחלקת ההפקה של Macmillan. מוציאים לאור אחרים ישתמשו בתבניות סגנון אחרות, לפי רצונם.

במסמך באנגלית, כותרת מרמה C מצוינת על ידי הסימון (c) ומותאמת לסגנון HC Word, כמוצג בתרשים 2.1.




```
on to your hat, we're going to do this very quickly.

(c)Analysis and Design

With a large project such as EmployeeNet, I would expect to
invest a significant amount of time on Analysis
(understanding the requirements) and design (modeling the
solution) before beginning implementation (writing the
code).

In 1992 I would have speculated that EmployeeNet was a two
year project. I would have been wrong, of course; by the
time we were done, it would have slipped out to three years!

By 1995 I might have known that we had less than a year to
build a project like this, because the world is speeding up,
and Internet sites had better hit the market in under a
year.

The tools are getting better, we're all getting smarter
about how to build Web applications and the competition is
fierce. The time frame keeps shrinking. Today, I believe we
must endeavor to get a first release out the door in 4-6
months. Perhaps less. To do this, I'd spend about a month on
analysis, another month on design and then two months coding
```

תרשים 2.1: דוגמה של כותרת מרמה C.

כשאני כותב את החומר, אני גם מסמן קטעים מסוימים לטיפול מיוחד על ידי העורכים. הסכמתי אתם על סימון מיוחד, בשם "**מונח טכני**", המספק הגדרות של מונחים טכניים. כשברצוני לציין "**מונח טכני**" אני מקליד: ***‏**מונח טכני**‏***

זוהי הנחיית הוצאה לאור עבור העורך, המציינת כי השורות שבין הכוכביות אמורות להופיע לצד סימון "**מונח טכני**". להלן דוגמה:



הנחייה למו"ל (Publishing Directive) היא סימול המשמש לציון טיפול מיוחד הנדרש ממחלקת ההפקה.

"**מונח טכני**" מסומן גם באחד משני סגנונות מיוחדים: PD (Publishing Directive) או NO (Note). אלה משמשים לסרגלי צד (Sidebars), כגון: הערה, "**מונח טכני**", "כיצד מבטאים זאת?", "טיפ" וכן הלאה.

דוגמה נוספת: כשיש לי רשימת תבליטים, מבקש המו"ל שלי שאסמן כל נקודת תבליט ב- [b]. בעת העריכה סימול זה יוחלף בתבליט. אני גם משתמש בשני סגנונות. כל שורה מסומנת עם הסגנון BL (Bulleted List) מלבד השורה האחרונה, המסומנת עם BX. BX מציין שזהו התבליט האחרון, ולכן נדרש ריווח אופקי נוסף לאחר שורה זו.

הסגנון AU (Author) משמש לתת הכותרת של המחבר או לציון הכרת תודה. הסגנון FT משמש לטקסט הרגיל לאורך המסמך.

בהמשך, נעסוק במיוחד בקוד המקור, המסומן בשלושה סגנונות. שורה בודדת של קוד מצוינת על ידי הסגנון C1. וקוד של מספר שורות מצוין על ידי הסגנון C2, כאשר השורה האחרונה מסומנת על ידי סגנון CX. אלה יהפכו חשובים במיוחד כאשר נהפוך את המסמכים הללו למבנה ה-canoncial XML שלנו.

חשוב לציין: כל הסגנונות והסימולים המיוחדים שהזכרנו עד כה (ויש רבים נוספים שלא הזכרנו) הם ייחודיים להוצאת Macmillan. הם **אינם** חלק מ-XML, אלא ייחודיים לתחום שבו אנו עוסקים.

אם כך, עולה השאלה, מדוע אנו טורחים לכלול את רמת הפירוט הזו אודות תחום העיסוק (הוצאת Macmillan), כשאנו עשויים שלא להזדקק לה שוב לעולם?! התשובה היא, שלמרות ש**פרטי** גיליון הסגנונות הם ייחודיים ל-Macmillan, הנוהל הכולל המקיף לקיחת מסמך עם קבוצה של ציונים, סימולים, סגנונות ומטה-נתונים והפיכתם ל-XML – היא מטלה שכיחה למדי.

הסבה ל-HTML

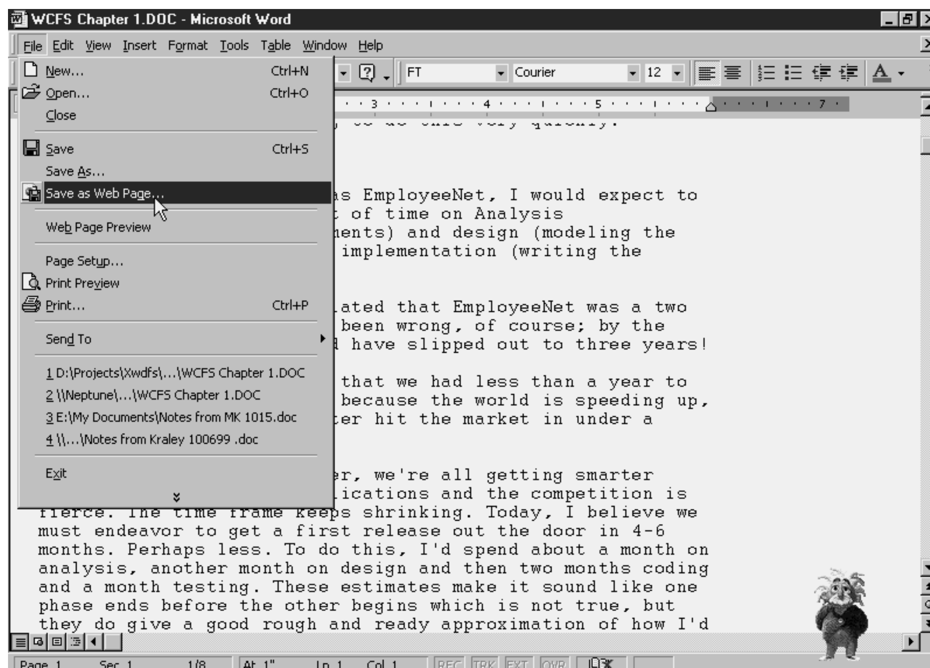
אם כך, אנו רואים שישנם שני נושאים שלובים. האחד הוא המרת מסמך ממבנה Word למבנה HTML, והשני הוא הבנת וקבלת מטה-נתונים המצויים כרגע בסימולי העריכה של Macmillan (כמו "Geek Speak", למשל) ובשימוש בסגנונות של Word.

בסופו של דבר איננו זקוקים לקבל את הסגנונות. לא מעניין אותנו ששורה מסוימת סומנה בסגנון HA. עם זאת, כן איכפת לנו לקבל את המטה-נתונים הניתנים כרגע על ידי אותו סגנון. כלומר, בעוד שלא איכפת לנו שזה היה סגנון HA, כן איכפת לנו שזו היתה כותרת מרמה A. מה שאנו עושים למסמכי Macmillan, עשינו קודם, ונמשיך לעשות, ללקוחות רבים נוספים; רק הפרטים משתנים.

מ-Word ל-HTML

מעבד התמלילים Microsoft Word מאחסן את מסמכיו בתצורה בינארית מוצפנת ייחודית. אני מניח שניתן לכתוב תוכנית מותאמת שתקרא את התצורה הבינארית ישירות, אך זה עשוי להיות קשה ומסורבל. לחילופין, ניתן להשתמש באוטומציית COM בכדי להתקשר עם מודל האובייקט Word ולשלוט במסמך דרך ממשק COM. זוהי גם כן מטלה מורכבת וקשה. יתרה מזאת, שתי הטכניקות אינן נחוצות; ניתן לשמור את המסמך במבנה HTML ואז לשלוט בו ביתר קלות.

בכדי לעשות זאת, פתח את Word והשתמש באפשרות **שמירה כדף אינטרנט** (Save As Web Page) מתוך התפריט **קובץ** (File), כמוצג בתרשים 2.2.



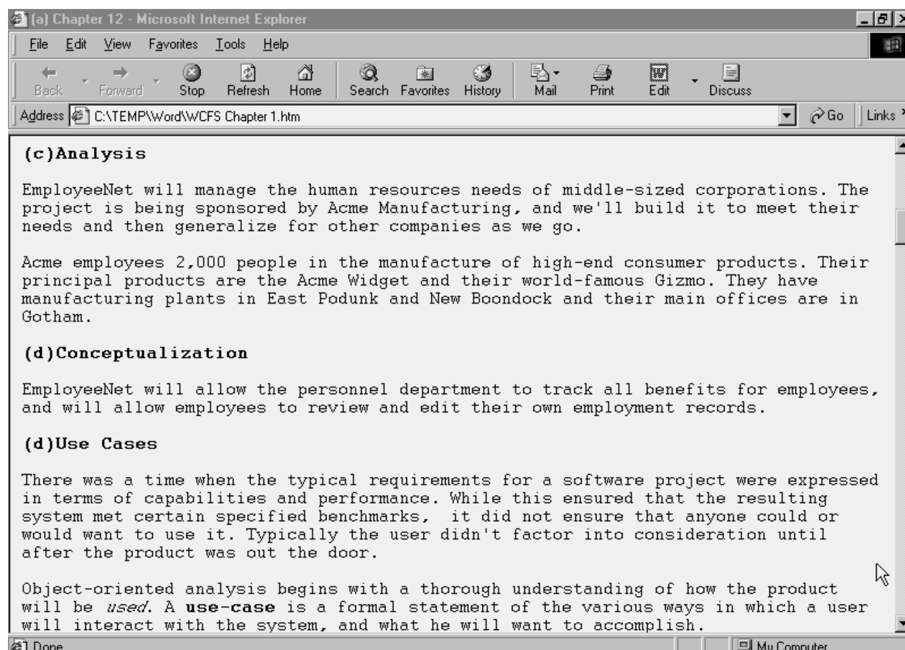
תרשים 2.2: שמירת המסמך כדף אינטרנט



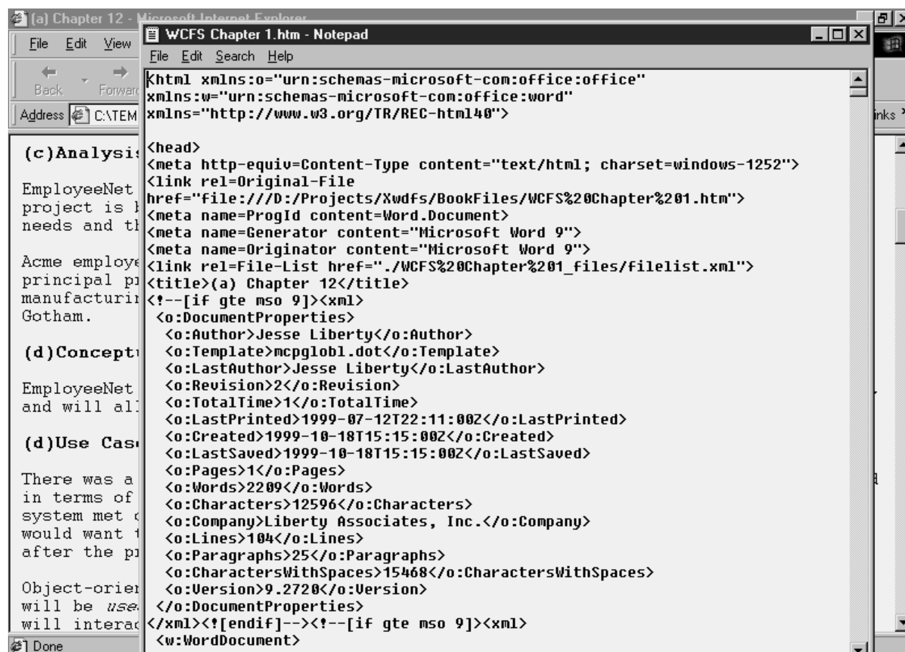
אם אינך רואה את האפשרות "**שמירה כדף אינטרנט**" בתפריט "קובץ" במעבד ה- Word שלך, וודא תחילה שהתקנת את Word 2000 (גרסאות מוקדמות יותר לא תפעלנה בגירסה זו), ואז נסה להרחיב את כל התפריט, שכן אפשרות זו מוסתרת לעתים קרובות.

אם אין ברשותך את Word 2000, השתמש במסמכים המומרים מראש עם סיומת htm. המצויים בדיסק, או הורד אותם מהאתר בכתובת: www.LibertyAssociates.com.

הבה נעיף מבט בתוצאה על ידי בחירה ב-File > Web Page Preview (קובץ > תצוגה מקדימה כדף אינטרנט), כמוצג בתרשים 2.3.



תרשים 2.3: צפייה במסמך ה-HTML החדש.



תרשים 2.4: האפשרות View > Source נבחרה.

ניתן לראות בתרשים 2.3 את הקובץ בדפדפן, ובעוד שחלק מהסגנון המפורט עשוי להשתנות קלות, התוצאה נראית קרובה למדי למה שהיינו מצפים. השינויים החשובים שמתחת לפני השטח נחשפים כאשר נבחר באפשרות View < Source (תצוגה < מקור) כמוצג בתרשים 2.4.

מה כל זה? זה לא נראה כלל כמו מה שהיינו מצפים.

בגירסאות Word שקדמו ל- 2000 ניתן לשמור מסמך במבנה HTML, אך היה זה תהליך מאכזב למדי. חלק ניכר מן התצורה (למשל, טורים, הזחת שורות תחיליות, מספור, מיתאר וכן הלאה) אבד במעבר מקובץ ה Word הבינארי לקובץ המתורגם ל-HTML. מפריעה באותה מידה היא העובדה ששמירה ל-HTML היתה מעבר חד-כיווני. לא היתה כל דרך קלה לחזור ל-Word; נתוני מבנה המסמך אבדו ללא כל אפשרות לשחזרם.

ב- Office 2000 השתמשה מיקרוסופט ב-XML כחלופה לתצורה הבינארית הייחודית של המסמכים. האפקט הראשון של המעבר ל-XML הוא שמסמך האינטרנט המיוצר ב-Word מכיל עתה את כל נתוני המבנה, ויש נאמנות למקור במידה גדולה הרבה יותר בין מסמך HTML המתקבל לבין מסמך Word המקורי. מכיון שנתוני המבנה (מטה-נתונים) נשמרים, ניתן עתה לבצע שינויים במסמך HTML, ולחזור ולשמור אותו כמסמך Word וכך לאחזר את המבנה הייחודי של Word.

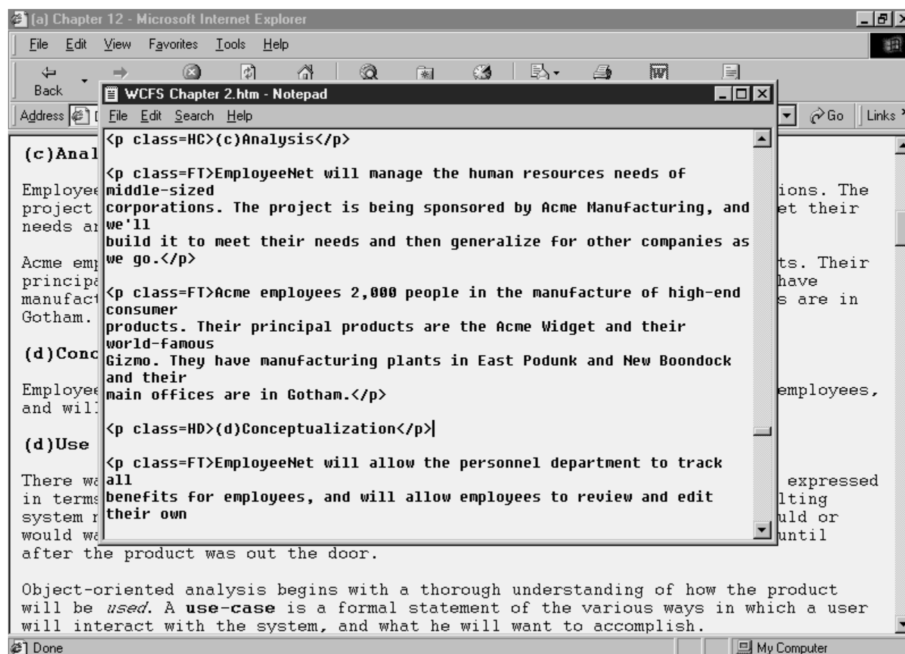


> עם זאת ראוי לציין, כי למרות שמיקרוסופט אמנם משתמשת ב-XML לאחסון מסמכים, מבנה XML המתקבל לאחר שמירת קובץ Word אינו מעוצב היטב, והמסמך מכיל חלקים רבים שאינם XML כלל. המצב המושלם יהיה ביום בו מסמכי Word יישמרו כ-XML מעוצב היטב, כך שנוכל בקלות לשנות את ייעודם למדיות שונות, ללא כל צורך בתיכנות נוסף. גם היום הזה אינו רחוק.

בתרשים 2.4 אנו יכולים לראות אי של XML (XML Island), המכיל את התכונות של המסמך (למשל, היסטוריית שינויים, מספר תווים וכן הלאה). נדלג על אי XML זה כעת ונגלול לתחילת המסמך שלנו, כמוצג בתרשים 2.5.

אי XML (XML Island) הוא קטע XML המוכל בתוך מסמך HTML. זו הרחבה של מיקרוסופט ל-HTML, ההופך את העבודה עם XML לקלה ופשוטה הרבה יותר.

כעת מצבנו טוב יותר. כאן אנו רואים קבוצה מקובלת של משפטי HTML. כל פיסקה מסומנת עם תגית <p>, ורובן משתמשות בתכונות המחלקה CSS כדי לשלוט בסגנון מסוים.



תרשים 2.5: גלילה מטה בקוד המקור.

סיור קצר

HTML: התגית <p> מציינת פסקה ב-HTML. נשתמש בתכונת המחלקה יחד עם CSS (Cascading Style Sheets) כדי לשלוט בעיצוב תוכן הפיסקה.

בהמשך לכך, שים לב שהשורה השנייה למעלה מראה: <p class=FT>

אנו יכולים לראות שמחלקה זו הוגדרה כבר בקובץ. חפש את ההערה ".FT" מראש הקובץ (שים לב לנקודה [.]). הדבר יוביל לקוד הבא:

```
p. FT, li.FT, div.FT
    {mso - style - name:FT;
    mso - style - parent:" " ;
    margin - top:0in;
    margin - right:0in;
    margin - bottom:12.0pt
    margin - left:0in;
    line - hieght:12.0pt
    mso - pagination:widow-orphan;
    font - size:12.0pt
    mso - bidi - font - size:10.0pt;
    font - family:courier
    mso - fareast - font - family:"Times New Roman";
    mso - bidi - font - family:"Times New Roman";}
```

קוד זה הוא בתוך קטע גיליון עיצוב (לא מוצג). השורה הראשונה מציינת שכשנעשה שימוש במחלקה במסגרת תגיות <p>, <li> או <div>, היא תאופיין על ידי הסגנון הנלווה. הסגנון המוגדר מתאים לסגנון mso (Microsoft Office) הקרוי FT. והוא בעל המאפיינים שהוצגו, ובכללם גובה שורה של 12.0 נקודות, משפחת גופנים של Courier וכן הלאה.

הציון class בתגית <p> המוצגת בתרשים 2.5 מנחה את הדפדפן להחיל את הסגנון המצוין על פיסקה זו. אנו נדבר יותר על סגנונות CSS ככל שנתקדם.

HTML בנוי היטב

התאחדות רשת האינטרנט העולמית W3C (World Wide Web Consortium), הכריזה רשמית על סדרת המלצות ל-HTML. המלצות אלו יוצרות תקן בינלאומי עבור HTML משופר, או בנוי היטב (Well-Formed HTML). זהו תקן גמיש למדי, כמו שגם השפה HTML היא מאוד סלחנית. היא תוכננה לכתיבה ידנית (הנעשית לרוב ב-Notepad!) ולכן, למשל, תגיות HTML רבות שנפתחות אינן מחייבות תגית סגירה. כתיבת תגית <p> ללא תגית סגירה </p> מתקבלת על ידי HTML כמעשה חוקי לגמרי. HTML סלחנית במקרים נוספים רבים, למשל, אין חובה להציג ערכים של מאפיינים בתוך גרשיים. הקוד הבא, מוכר על ידי הדפדפן: <h1 align=right>.

XML היא בדיוק ההיפך, שפה זו תוכננה להיות קשיחה מאוד. למה? כי באמצעות שפה נוקשה, קל יותר ליצור כלים: ככלות הכל, יש צורך להתמודד עם פחות חריגות.

אחד הקשיים שיש לתלמידי HTML מתחילים, מתבטא בחוסר היכולת להגדיר במדויק מה מותר ומה אסור ב-HTML. המרצה טוען שתגית מסוימת חייבת להיסגר, ואילו התלמיד מגלה שגם כשאותה תגית אינה סגורה הכל עובד.

הנה הסיבה לכך שאני אוהב ספרדית. ברגע שאתה לומד את כללי ההיגוי, אין כל שאלה לגבי אופן ההיגוי לכל מילה בשפה (כמעט לכל מילה). כשאתה נתקל במילה חדשה, למשל שם עיר כמו Ocho Rios, אינך צריך לנחש. Ocho חייב להיות מבוטא כ-Och-oh (מתחרז עם coach-oh) ו-Rios כ-Ree-ose (מתחרז עם me-oh, עם s שורקת בסוף). שפה נוקשה הופכת את החיים לקלים. השווה זאת לצרפתית, למשל.

שפה גמישה כמו אנגלית עשויה לשגע אותך. חבר ישראלי שאל אותי פעם, "למה ל-"digit" יש g רכה ול-"dig it" יש g קשה?" כנראה שיש לכך תשובה טובה, אך לכל כלל באנגלית, יש לפחות יוצא מן הכלל אחד.

נוקשותה של שפת XML היא מעלתה. הרבה יותר קל לבנות כלי XML מאשר כלי HTML, מכיון ש-XML היא הרבה יותר צפויה. לרוע המזל, במקרה שלנו כאן, יש לנו בעיה, מכיון ששמירת מסמך כדף אינטרנט מתוך Word יוצרת HTML תקני, והסבה שלו ל-XML עלולה להיות מסובכת.

לכן, הדרך הטובה ביותר להתחיל היא להעביר את מסמך HTML הסטנדרטי למסמך XHTML – כלומר, למסמך HTML המותאם ל-XML.

XHTML הינה שפה מיוחדת, אשר באה ליצור את המעבר בין HTML הסטנדרטי והסלחן ל XML. זהו שילוב של HTML ו-XML. למעשה, כשתקרא קוד XHTML תגלה שהוא דומה מאוד לקוד HTML. ההבדל הוא שקוד זה עומד בדרישות החוקיות של HTML, ו-XML גם יחד.

לרוע המזל, ההמלצה ל-XHTML התקני נמצאת עדיין בטיוטה, אך אין זה צריך לעצור בעדנו. אנו יכולים לשכתב את מסמך HTML על פי הכללים של XML הבנוי היטב. זה עדיין יהיה HTML חוקי, אך נדרוש ממנו לעמוד בקווים המנחים הנוקשים יותר של XML. זו התמצית של XHTML. כך נוכל להרגיש בטוחים שהמוצר שנקבל יהיה עקבי עם ההמלצה הסופית של W3C ל-XHTML.

להלן נושאי מפתח בהפיכת מסמך HTML למסמךXML הבנוי היטב:

- אסור שתגיות מקוננות יחפפו.
- שמות התגיות חייבים להיות תלויי רישיות (Case-Sensitive).
- כל התגיות חייבות להיסגר.
- ערכי תכונות (Attributes) חייבים להיות מסומנים בגרשיים.
- המסמך חייב להכיל רכיב אחד בדיוק, שיהווה את רכיב השורש של המסמך.

בנוסף, כלל שלוש קובע כי כל התגיות חייבות להיסגר על ידי תגית סוגרת. עם זאת, ייתכנו מקרים בהם נרצה להשתמש בתגיות בודדות, אשר אינן מכילות טקסט פנימי. ב-HTML התגית
 היא דוגמה לתגיות מסוג זה. תגיות אלו חייבות להיות מיוצגות ב-XML בנוי היטב על ידי לוכסן לפני שם התגית. לכן, התגית
 ב XHTML צריכה להיראות כך:
 (שים לב לרווח לפני הלוכסן -זהו מנהג מקובל, אך אינו חובה).

מסתבר שחוק מספר אחד ("אסור שתגיות מקוננות יחפפו") מיושם די טוב בהמרה מ-Word ל-HTML. כך גם עם "שמות תגיות תלויי רישיות". לרוע המזל, לא כל התגיות נסגרות, ולא כל ערכי התכונות מופיעים בגרשיים - דבר הדורש תיקון ידני. .

תיקון מסמך HTML

יש מספר גישות שנוכל לבחור בהן לתיקון מסמך HTML. נוכל, למשל, לתקן אותו ידנית, אך קרוב לוודאי שפעולה כזו תעייף אותנו במהרה במסמכים גדולים.

חלופה אחרת היא להשתמש בביטויים מוסכמים ובתוכנית לניתוח תחבירי (Parser) לתיקון הדברים, אך גם זה עשוי להסתכם בעבודה רבה. HTML אינה שפה מחמירה או נוקשה (וגם אינה רגולרית), וכמו באנגלית יש בה חריגות רבות מן הכללים.

הציטוט הבא הוא מתקציר של דברים שכתב לי חבר לגבי בעיה זו:

> מה שאנו באמת זקוקים לו זה נַתָח תחבירי (Parser) של HTML היודע את החוקים הללו והחריגות מהם, ואז נוכל להפיק מסמך הבנוי היטב. אנו יכולים לבנות נתח כזה, או שנוכל לחפש ברחבי האינטרנט כדי לראות אם יש למישהו תוכנית כזו למכירה או שימוש.

אך חכה רגע, יש לנו נתח תחבירי מעולה עבור HTML בדפדפן שלנו. MSHTML הוא ה-DOM של HTML שניתן לנו על ידי מיקרוסופט. זהו רכיב המבצע את מירב הניתוח של קובץ HTML והסבתו לעץ DOM, שבו אנו שולטים בקלות בכתיבת תסריטים (Scripting) בצד הלקוח.

על ידי התייחסות ל-MSHTML ול-Internet Explorer ב-VB, אנו יכולים לכתוב בקלות רכיב המבצע את מה שאנו רוצים, ומבטיח להתמודד עם כל קובץ HTML ש-IE יכול לקרוא. ומכיוון שאנו יוצרים את הפלט, גם נוכל להבטיח שיהיה בנוי היטב.

הנה כי כן, נוכל להשתמש בתוצר של הנתח התחבירי של HTML המובנה בתוך IE. מכיון שמיקרוסופט מציעה את תבנית הניתוח הזו כאובייקט ActiveX עצמאי (MSHTML.DLL) יש לנו גישה מלאה ליכולותיו.

בהמשך הלימוד בפרק זה נלמד כיצד לעשות זאת. עם זאת, לפני שנתחיל, הרשו לי לסקור משפט אחד מהמכתב של חברי שהצגתי לעיל: "MSHTML הוא ה-DOM של HTML שניתן לנו על ידי מיקרוסופט. זהו רכיב המבצע את מירב הניתוח של קובץ HTML והסבתו לעץ DOM, שבו קל לנו לשלוט על ידי כתיבת תסריטים (Scripting) בצד הלקוח".
הבה נפשט משפט זה. MSHTML הוא ה-DLL בו אנו משתמשים כדי לגשת ל-DOM של MSHTML. DOM זה הוא המאפשר את DHTML. DHTML היא, בתמצית, לא יותר מאשר מניפולציה של ה-DOM של HTML, בדרך כלל על ידי קוד תסריט. קוד התסריט נכתב לרוב ב-JavaScript, אך כמובן שאינו חייב להיות כזה: ה-DOM של HTML אינו תלוי-שפה.

כאשר אנו עובדים עם DHTML, אנו לוקחים את ה-DOM של HTML כמובן מאליו: הוא מייצג את מסמך HTML ואנו שולטים בו ישירות בתסריט שלנו. מה שחברי מציע כאן, הוא שנקרא את מסמך HTML הנשמר מתוך Word אל ה-DOM של HTML המסופק על ידי MSHTML. כך תהיה לנו גישה לכל המרכיבים של המסמך דרך ה-DOM.

כאן עולים שני נושאים אפשריים שעלולים לגרום לנו בלבול: ההבדל בין המסמך וה-DOM, וההבדל בין ה-DOM של HTML ובין ה-DOM של XML.

מסמכים ו-DOMs

מודל האובייקטים של המסמך של HTML, (Document Object Model (HTML DOM)), הוא מפרט (Specification) לאופן בו מסמכי HTML בנויים. כאשר אנו יוצרים מופע של ה-DOM, בפועל אנו יוצרים מודל מופשט של מסמך ה-HTML במרחב הזיכרון של המחשב.

עד לאחרונה, כל דפדפן היה חופשי ליצור מודל משלו של המסמך, ו-Internet Explorer ו-Netscape יצרו מודלים שונים. בימים המוקדמים (לפני שנה או שנתיים), אף אחת משתי החברות לא הפכה את המודל שלה לזמין למתכנתי האינטרנט העתידים לבוא, כך שזה לא ממש היה חשוב כיצד המודל עבד; ככלות הכל, זה היה מידע חבוי ושמור בזכויות יוצרים.

במאמץ להפוך את מודל האובייקטים של המסמך לסטנדרטי, קבוצת עבודה ב-W3C היתה אחראית ליצירת המלצות למודל מסמך HTML המוסכם על כולם ברחבי העולם. המטרה היתה לגרום לכל הדפדפנים לתמוך במודל זה, ולחשוף את האובייקטים בתוך המודל לשליטה מתוך התסריטים (Scripts).

ה**אובייקטים** בתוך המודל הם האלמנטים הנשלטים ביישום DHTML. למשל, אתה עשוי לרצות לשלוט בסגנון או בתוכן של טקסט, לשנות כותרות, או לשנות פרטי תצוגה אחרים. בנוסף, אתה עשוי לרצות לגשת לתוכן של שדות טקסט או תיבות-רשימה (List Boxes) בטופס. החלון, הטופס, שדות הטקסט והתמונות הם כולם אובייקטים בתוך מודל האובייקטים. כשאתה מתכנת סקריפט לשינוי תמונה במעבר עכבר, אתה משנה את מאפיין המקור (SRC) של אובייקט התמונה המסוימת, עליה מועבר העכבר.

DOMs כנגד DOMs

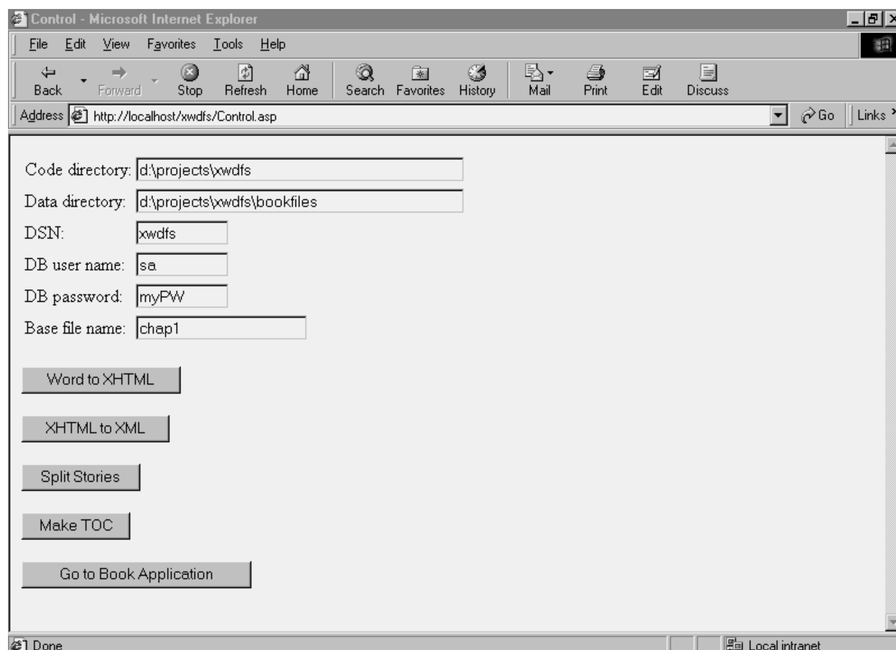
התחום השני של בלבול אפשרי הוא בין מודל האובייקטים של מסמך ה-HTML ובין מודל האובייקטים של מסמך ה-XML. MSHTML מספק גישה ל-DOM של HTML, ולכן שליטה באובייקטי ה-HTML, אך אינו יודע דבר אודות XML. אנו נראה את ה-DOM של XML בפעולה בפרקים מאוחרים יותר. לעת עתה, ה- DOM בו נשלוט יהיה ה-DOM של HTML.

המרת Word ל-XHTML

בכדי להשיג את מטלת ההמרה של מסמך HTML זה, אשר נוצר על ידי Word, ל-XHTML, אנו נבנה אובייקט ActiveX. ב-VB זה פשוט כמו יצירת פרויקט חדש DLL ActiveX.

הרעיון הוא שה-DLL יכיל בתוכו ארבעה אובייקטים: Word2XHTML, WordXHTML2XML, XSLTransform ו-SplitStories. כרגע, היחידי שמעניין אותנו הוא Word2XHTML. כפי שהשם מרמז, זהו האובייקט שימיר את קבצינו מ-HTML של Word ל-XHTML.

כל זה נשלט על ידי Control.asp, כמוצג בתרשים 2.6.



תרשים 2.6: Control.asp.

זהו קובץ ה-ASP היחידי שאנו נשתמש בו במהלך הפרויקט כולו. מכיוון שברצוננו להתמקד ב-XML, קובץ ASP זה הוא באמת יצירה מהירה המשמשת לאיסוף המידע שאנו צריכים (ספריות, סיסמאות וכן הלאה) ושיגור אובייקטי ה-ActiveX שלנו. אנו נבחן את ה-ASP ככל שנתקדם, אך אנו לא נתעכב עליו.

סיור קצר

הבה נעיף מבט מהיר לראות מה נדרש כדי לשגר את Word2XHTML, הרכיב הראשון שלנו, שתפקידו להמיר את ה-HTML המיוצר על ידי Word ל-XHTML.

אנו נבחן עתה כיצד דף ה-ASP עובד וכיצד הוא מפעיל את Word2XHTML, ואנו נשוב לשארית קובץ ה-ASP הזה ככל שנתקדם, כדי לראות כיצד הוא מתקשר עם שאר הרכיבים ביישום שלנו.

אינך זקוק לעקוב אחר משהו מכל זה כדי להמשיך בעבודתנו על XML. אתה יכול פשוט להשתמש בדף ה-ASP שנמצא בתקליטור המצורף. כמובן, אתה עדיין צריך להתקין את מסד הנתונים שבתקליטור, ועליך להתאים חיבור DSN למסד נתונים זה, כמוסבר בהמשך. תחילת control.asp נראית בתדפיס 2.2.

```
<%
Option Explicit
Dim DBConn, rs, fso, DSN, DBUser, DBPass, codeDir, dataDir, baseName, dataPath

'some of our operations may take a long time
Server.ScriptTimeout = 3000

'if one of our command buttons was pressed, store the params as cookie
if Request("Cmd") <> "" then
        SetCookieVal "CodeDir"
        SetCookieVal "DataDir"
        SetCookieVal "DSN"
        SetCookieVal "DBUser"
        SetCookieVal "DBPass"
        SetCookieVal "BaseName"
        'expire in a year
        response.cookies("XWDFS").Expires = dateadd("yyyy", 1, date)
end if

'retrieve the params from the cookie
codeDir = GetInput("CodeDir")
dataDir = GetInput("DataDir")
DSN = GetInput("DSN")
DBUser = GetInput("DBUser")
DBPass = GetInput("DBPass")
baseName = GetInput("BaseName")


'we almost always need a database connection
set DBConn = Server.CreateObject("ADODB.Connection")

if DSN <> "" and DBUser <> ""  and codeDir <> "" and dataDir <>
➥ "" and baseName <> "" then

        DBConn.Open DSN, DBUser, DBPass

        'and also a File System Object to manipulate directories
        set fso = Server.CreateObject("Scripting.FileSystemObject")

        dataPath = fso.BuildPath(dataDir, baseName)

        'now perform the requested actions

```

```
		select case Request("Cmd")
			case "Word to XHTML"
				Response.Write Word2XHTML()

			case "XHTML to XML"
				Response.Write XHTML2XML()

			case "Split Stories"
				Response.Write SplitStories()

			case "Make TOC"
				Response.Write MakeTOC()

			case "Go to Book Application"
				Response.redirect "book.htm"

			case "Show Chapter as HTML"
				Response.Write ShowHTML()

			case "Show Chapter as XML"
				Response.Write ShowXML()

		end select
	else
		Response.write("Please fill in all required fields before proceeding.")
	end if
%>

<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" content="text/html; charset=iso-8859-1">
<TITLE>Control</TITLE>
</HEAD>
<BODY>
<form>
<table>
<tr><td>Code directory:</td><td><input size=40 name=CodeDir
➥value="<% =CodeDir  %>"></td></tr>
<tr><td>Data directory:</td><td><input size=40 name=DataDir
➥value="<% =DataDir  %>"></td></tr>
<tr><td>DSN:</td><td><input size=10 name=DSN
➥value="<% =DSN  %>"></td></tr>
<tr><td>DB user name:</td><td><input size=10 name=DBUser
➥value="<% =DBUser  %>"></td></tr>
```

```
<tr><td>DB password:</td><td><input size=10 name=DBPass
➥value="<% =DBPass  %>"></td></tr>
<tr><td>Base file name:</td><td><input size=20 name=BaseName
➥value="<% =BaseName  %>"></td></tr>
</table>
<p><input type=submit value="Word to XHTML" name=Cmd>
<p><input type=submit value="XHTML to XML" name=Cmd>
<p><input type=submit value="Split Stories" name=Cmd>
<p><input type=submit value="Make TOC" name=Cmd>
<p><input type=submit value="Show Chapter as XML" name=Cmd>
<p><input type=submit value="Show Chapter as HTML" name=Cmd>
<p><input type=submit value="Go to Book Application" name=Cmd>
</form>
</BODY>
</HTML>
<%
 Function GetInput(name)
        GetInput = Request.Cookies("XWDFS")(name)
 End Function

 Sub SetCookieVal(name)
        Response.Cookies("XWDFS")(name) = Request(name)
 End Sub

 Function Word2XHTML()
        'convert a Word .htm file to XHTML
                dim oW2X, inFile, outFile

        set oW2X = Server.CreateObject("FromScratch.Word2XHTML")

        'the input file from Word should have a .htm extension
        inFile = dataPath & ".htm"

        'we'll save the output in a .xhtml file
        outFile = dataPath & ".xhtml"

        oW2X.ConvertToXHTML inFile, outFile
        Word2XHTML = "Converted " & inFile & " to " & outFile
 End Function
```

בשורה 0 של control.asp אנו פותחים תסריט צד-שרת עם הסימן %<, המורה לדפדפן שמה שיבוא בהמשך הינו קוד ASP שירוץ על השרת (Server).

בשורה 1, Option Explicit מנחה את מתרגם ה-VBScript להתלונן אם נעשה שימוש במשתנים שלא הוגדרו תחילה. אפשרות זו מצילה אותנו משגיאות מייגעות הנגרמות משגיאות איות של שמות משתנים.

אנו מצהירים מספר משתנים בשורה 2 ; אנו נשתמש בהם לאורך קובץ ה-ASP.

בשורה 5 אנו מכוונים את משתנה Timeout ל- 50 דקות, שזהו זמן ארוך, אולי עד כדי גיחוך, הארוך הרבה יותר מהפעולה הארוכה ביותר שנבצע. תוכל להוריד זמן זה ל- 10 או 15 דקות, כרצונך ; העיקר שיהיה מספיק זמן כדי להבטיח שלא תחרוג מהזמן אלא אם כן היישום תקוע ללא תקווה.

מרבית שארית התסריט מוקדשת לניהול העוגיות (Cookies) שיאחסנו את משתנינו. אנו צריכים לשאול את המשתמש היכן מצויים מסמכי ה-HTML, היכן מודולי הקוד שלנו, וכיצד להתקשר עם מסד הנתונים. ברגע שנקבל ערכים אלו, נאחסן אותם בתוך עוגיה במחשב הלקוח, כך שהמשתמש לא יצטרך להכניס אותן שוב.



אם אינך מכיר cookies או ASP, אנא ראה את רשימת הקריאה בנספח ד'. אתה יכול לדלג על כל ההסבר אם אתה רוצה, ולהמשיך ב-XML, ולהתייחס בינתיים לקובץ ה-ASP השולט כקסם - אם כי ידע בכתיבת ASP הינו חשוב לעבודה, ומומלץ מאוד ללא שום קשר.

שורות 8-17 עוסקות בהגדרת העוגיה, ושורות 19-25 עוסקות בהשגת הערכים מהעוגיה. ממשק המשתמש הממשי להשגת ערכים אלו מהמשתמש מוראה בשורות 70 עד 94.



אם אינך מכיר ADO, אנא ראה את רשימת הקריאה בנספח ד'. אתה אמור להיות מסוגל לעבור את ההסבר הזה ללא ידע רב ב-ADO, ושוב, אתה יכול פשוט לדלג על חלק זה ולהשתמש בדף ה-ASP שבתקליטור המצורף.

בשורה 29 אנו יוצרים אובייקט חיבור ADO אשר יעניק לנו את יכולת ההתקשרות עם מסד הנתונים. בשורה 31 אנו מוודאים שבידינו הערכים הנדרשים להתקשרות עם מסד הנתונים, ואם כן, בשורה 33 אנו פותחים את ההתקשרות, תוך שימוש ב-DSN, שם המשתמש והסיסמה כפי שסופקו על ידי המשתמש.



מהו DSN?

ישנן בשוק מספר מערכות בסיסי נתונים מסחריות. בין הידועות, ודאי שמעת על: SQL Server, Oracle, Access ועוד. כל מערכת כזו מנהלת את המידע באמצעות שפה בינארית ייחודית לה. כדי לגשת אל בסיס הנתונים, על מנת לעדכנו, לשנותו, או לקרוא נתונים ממנו, יש לפנות אל בסיס הנתונים בשפה המוכרת לו. שפת השאילתות המובנית SQL, הינה השפה הידועה ביותר, הפשוטה, והמקובלת בכתיבת שאילתות לבסיסי נתונים.

לכן, כדי לפנות לכל בסיס נתונים באמצעות התחביר המוכר של SQL, יש צורך במתרגם (הקרוי דרייבר) אשר יעשה את ההמרה בין פקודות ה-SQL לשפה הבינארית הייחודית של אותו בסיס נתונים. ואכן, לשם כך הומצאה חבילת דרייברים שכאלה, הקרויה ODBC. זוהי חבילת דרייברים אשר מכילה מספר מתרגמים לבסיסי נתונים שונים הקיימים בשוק. כדי לפנות אל בסיס הנתונים דרך הדרייבר הזה, יש צורך ליצור חיבור DSN, אשר מורה לתוכנית הפונה (במקרה שלנו זו תוכנית ה-VB, במקרים אחרים זו יכולה להיות תוכנית אחרת שנכתבה ב- ++C, או בתוך עמוד ASP.) לגשת אל בסיס הנתונים דרך דרייבר נבחר, אשר ידע כיצד לתרגם את פקודות ה-SQL לפקודות המוכרות לבסיס הנתונים המבוקש.

אם כן, כדי להתקשר למסד הנתונים מ-VB, אתה חייב ליצור מקשר DSN שייקשר למסד הנתונים של ה-XML. להלן הדרך לעשות זאת:

1. פתח את לוח הבקרה שלך ולחץ על "ODBC Data Sources". אם אתה משתמש ב-Windows 2000, תמצא סמל זה בתיקיה Administrative Tools, שבתפריט Programs.
2. בחר בלשונית "System DSN" ולחץ "Add".
3. בחר ב-"SQL Server" מתוך רשימת הכוננים ואז לחץ "Finish".
4. בשדה "Name" הכנס שם – אני מציע "XML". בשדה "Description" הכנס תיאור אם אתה רוצה (למשל, "גישה למסד הנתונים הניסויי של XML") ובשדה "Server" בחר בשרת. אתה אמור לראות את השרת שעליו אתה מריץ את SQL Server (או local, אם אתה מריץ את SQL Server מהמחשב המקומי). אם לא, פנה למנהל המערכת שלך. לחץ על "Next".
5. לחץ על "...With SQL Server authentication" כשאתה משתמש בשם המשתמש ובסיסמה שהוכנסה בחלון ה-Properties של SQL Server. מחק את administrator מהשדה "LoginID" והכנס את שם המשתמש והסיסמה הנכונים (הרבה אנשים משתמשים ב-sa ובסיסמה ריקה). לחץ "Next". ברגע זה, תחפש התוכנית אחר שרת SQL Server, על פי הנתונים שהזנת, השם והסיסמה. במידה והכל עובר בשלום, תועבר למסך הבא. במידה והניסיון נכשל, תתקבל הודעת התראה מתאימה והיישום ייפסק.
6. לחץ על "Change the Default Database" וגלול מטה לבחירת מסד הנתונים XWDFS. לחץ על "Next".
7. קבל את כל ברירות המחדל בתיבת הדיאלוג "Create A New Data Source to SQL Server", ואז לחץ על "Finish".
8. לחץ על "Test Data Source". אם המבחן לא הושלם בהצלחה, תזדקק לסיוע ממנהל ה-SQL Server שלך, מנהל המערכת, או Microsoft. בכל אופן, מרבית הסיכויים שתקבל את ההודעה

"Test Completed Successfully".

איחוליי, כעת יש לך מערכת DSN הקרויה XML. לחץ "OK". זה יחזיר אותך לתיבת הדיאלוג "ODBC Data Source Administrator", לחץ שוב "OK".

עם מסד הנתונים וה-DSN במקום, פקודת הפתיחה בשורה 33 יוצרת את ההתקשרות. בשורה 36 אנו פותחים קובץ אובייקט מערכת (File System Object) ומגדירים את הנתיב שלו לתיקיית המידע ושם הבסיס המסופקים על ידי המשתמש. מדובר יותר בהתקשרות לתיקיה שבה נעבוד.

זה מביא אותנו לשורה 42. כשאנו לוחצים על "Word to XHTML", שזוהי מטלתנו הראשונה, שורה 44 תפנה לפונקציה ()Word2XHTML ותדפיס לדפדפן את התוצאות.

Control מדלג מטה לשורה 104, במקום בו Word2XHTML מיושם. אנו מגדירים שלושה משתנים מקומיים: אחד כדי להחזיק את אובייקט ה-ActiveX בו נשתמש, שני להחזקת קובץ הקלט (קובץ ה-HTML שלנו), ושלישי להחזקת קובץ הפלט (קובץ ה-XHTML שלנו).

אנו קובעים את ערך המשתנה oW2X לאובייקט ה-ActiveX בשורה 108, ויוצרים את קבצי הפלט והקלט בהתבסס על נתיב המידע שהוקם קודם לכן.

לבסוף, בשורה 116, אנו קוראים לשיטה ConvertToXHTML באובייקט ה-ActiveX שלנו, בהעבירנו את שמות קבצי הפלט והקלט. כשאנו מסיימים, מוחזרת המחרוזת (לדוגמה) "Converted Chap1.htm to Chap1.xhtml" המודפסת לדפדפן.

בקטע הבא אתאר את אובייקט ה-ActiveX, שקראנו לו הרגע, בפירוט.

בתוך Word2XHTML

כזכור, מטרתנו היא להפוך את ה-HTML ל-XHTML) XHTML הוא מסמך XML בנוי היטב לכל דבר. העניין הוא שהוא מכיל תגיות HTML); כלומר, לוודא שתכונות מצויות במרכאות כפולות, תגיות אינן מקוננות וכן הלאה. יכולנו לכתוב תסריט Perl ולמצוא את כל השגיאות ולתקנן, אך זה יהיה מאמץ ארוך ומפרך. במקום זאת, ננצל את העובדה ש-IE5 יקרא בשמחה את ה-HTML הקיים ויספק לנו מודל אובייקטים למסמך.

לאחר שה-DOM נוצר בזיכרון, אנו יכולים בקלות לסרוק את ה-DOM, למצוא כל רכיב ורכיב, ולפלוט אותו לקובץ שלנו בתחביר XML מתאים.

כדי להתחיל, Control.asp קורא ל-ConvertToXHTML (שורה 116 של תדפיס 2.2), תוך העברת שם קובץ הקלט (קובץ ה-HTML שלנו שנוצר על ידי Word) ושם קובץ הפלט (קובץ HTML שהוא גם XML - בנוי היטב - או בקיצור XHTML), כמוצג בתדפיס 2.3.

תדפיס 2.3

```
0:  'Converts a HTML file so that the HTML is also well-formed XML
1:  'Attribute values must be quoted; tags must be closed and not overlap;
    ➥tags must agree in case
2:  'The approach is to read the document in to an HTML DOM,
    ➥and then output it in well-formed format
3:  Public Sub ConvertToXHTML(ByVal inPath As String, ByVal outPath As String)
4:      Dim ie As InternetExplorer, doc As MSHTML.HTMLDocument, url As String
5:
6:      Set ie = New InternetExplorer
7:
8:      'this may not work across machines
9:      url = "file://" & inPath
10:
11:     'open the HTML document
12:     ie.navigate url
13:
14:     'wait until the file finishes loading
15:     Do Until ie.readyState = READYSTATE_COMPLETE
16:         Sleep 1000
17:     Loop
18:
19:     'get the HTML DOM object
20:     Set doc = ie.document
21:
22:     'open our output file (overwriting if necessary)
23:     Open outPath For Output As #1
24:
25:     'and begin the recursive walk at the root
26:     OutputElement doc.documentElement, ""
27:
28:     Close #1
29:  End Sub
```

ניתוח

בשורה 3 קובץ הקלט וקובץ הפלט מועברים כפרמטרים. בשורה 4, משתנה מקומי, ie, מוגדר כ-InternetExplorer(1). בשורה 6, משתנה זה משויך לאובייקט InternetExplorer חדש.

אובייקט InternetExplorer זה מאפשר לנו לשלוט במופע של IE דרך ממשק אוטומציית ה-COM. בעוד שזה יהיה נושא מורכב מאוד ב- C++, הרי שב-VB זה יהיה כמעט ללא מאמץ. אתה פשוט יוצר מופע של אובייקט InternetExplorer חדש, כמוצג בשורה 6,

ואז קורא לתכונות ושיטות של האובייקט. לדוגמה, בשורה 20 אנו יכולים לגשת ישירות לתכונה Document (מסמך). התכונה Document מספקת לנו גישה ל-DOM של HTML.

בשורה 9 אנו יוצרים URL לקובץ הקלט ובשורה 17 אנו קוראים לשיטה navigate על אובייקט ה-Internet Explorer, בהעבירנו את אותו URL.

זה גורם למסמך להיטען לתוך אותו אובייקט של IE שיצרנו. ה-DOM של XML חושף את התכונה asynch. אם ערכו הוא true אז המסמך ינותח תחבירית (Parsed) במטלה (Thread) משלו. זה שימושי במקרה של דפדפן שיכול להמשיך גם במשימות אחרות במקביל, בעוד שהמסמך עובר תהליך של ניתוח. במקרה שלנו, אנו תמיד מציבים את הערך false, ובכך מאפשרים לפעולת הניתוח לחסום את התוכנית עד שהניתוח מסתיים.

ה-DOM של HTML אינו מציע את התכונה הזו; הוא תמיד אסינכרוני. אנו יכולים לכפות על התוכנית לעצור עד שהמסמך נטען במלואו, בכל אופן, על ידי השהיה עד ש-IE מסמן שה-readyState שלו הוא READYSTATE_COMPLETE, כמוצג בשורות 15-17.

נחזור לשורה 20, אשר כפי שראינו מקבלת את ה-HTML DOM, ואנו משייכים אותו ל-doc, שהגדרנו כיאות להיות מסוג MSHTML.HTMLDocument (הטיפוס של מיקרוסופט לאובייקטי HTML DOM).

בשורה 23 אנו פותחים את קובץ הפלט לפלט. זה יחזיק את מסמך ה-XHTML החדש שאנו ניצור. העבודה של שיטה זו היא, בכל אופן, בשורה 26, בה אנו קוראים ל-OutputElement, בהעבירנו את הרכיב המתקבל ממסמך ה-HTML שהשגנו בשורה 20, כמוצג בתדפיס 2.4.

תדפיס 2.4

```
0:  'Output the contents of the given element in well-formed XML format
1:  'Then recurse to el's children
2:  Private Sub OutputElement(el As Object, indent As String)
3:      Dim c, i As Integer, s As String, a As Object, n As Object
4:
5:      'open tag - note that MSHTML returns all tag names as uppercase,
        ➥so we don't have to worry about case matching
6:      'but I really like lower case tag names
7:      s = indent & "<" & LCase(el.nodeName)
8:
9:      'append all specified attributes, delimiting by space, and quoting the value
10:     'note that the attribute value may contain quotes,
        ➥so replace those with single quotes
11:     For Each a In el.Attributes
12:         'the attribute collection includes all possible attributes -
            ➥so we only care about those
13:         'that are "specified"
14:         If a.specified Then
```

```
            s = s & " " & a.nodeName & "=""" & Replace(a.nodeValue, """", "'") & """"
        End If
    Next

    'the style is another common attribute, but it is not include
    ➥in the attribute collections
    'note that the DOM likes to change the case of this as well,
    ➥so we normalize it to lowercase
    'also change embedded quotes to single quotes
    If el.Style.cssText <> "" Then s = s & " style=""" &
    ➥Replace(LCase(el.Style.cssText), """", "'") & """"

    'if we have no children, we can just end the tag here,
    ➥indicating it as an empty element
    If el.childNodes.length = 0 Then
        s = s & " />"
        Out s
        Exit Sub
    End If

    'otherwise just close the start tag
    s = s & ">"
    Out s

    'iterate thru the children
    For i = 0 To el.childNodes.length - 1
        Set n = el.childNodes(i)

        'In HTML, a node will either be an element or a text node
        If n.nodeType = 1 Then
            'an element - recurse
            OutputElement n, indent & "  "
        Else
            'text node, just output, but check first for special characters
            'the DOM cleverly converts entities to their reserved characters,
            ➥so we must convert them back again
            Out ConvertTextNode(n.nodeValue)
        End If
    Next

    'and finish with a close tag
    Out indent & "</" & LCase(el.nodeName) & ">"
End Sub
```

ניתוח

השיטה (Method) מתחילה בשורה 2. שים לב שכשאנו קוראים לשיטה זו אנו מעבירים שני פרמטרים, כמוצג בשורה 26 בתדפיס 2.3. הפרמטר הראשון, el, הוא רכיב; בפעם הראשונה הוא התכונה documentElement של האובייקט MSHTML.HTMLDocument. הפרמטר השני הוא מחרוזת בה נשתמש להזחת (Indention) כל שורה בפלט.

בפעם הראשונה, el הוא האובייקט documentElement, שהוא הפניה לצומת השורש של המסמך. זה בדיוק מה שאנו רוצים. אנו נתחיל בצומת השורש של מסמך הקלט, ואז נסרוק את העץ, בתהליך רקורסיבי על כל רכיב כדי לחלץ את מה שאנו צריכים כדי ליצור XHTML DOM, אותו נוכל אז לפלוט כקובץ XHTML.

בשורה 7 אנו מתחילים בעיבוד על ידי יצירת מחרוזת, אשר תצבור את מחרוזת הפלט. אנו מוסיפים תגית פותחת (<) ואז אנו מוסיפים את שם הרכיב. אנו משיגים את שם הרכיב על פי ה-tagName של הרכיב el, אותו אנו הופכים לאותיות קטנות (Lower Case).

כעת אנו צריכים לחזור באיטרציות על התכונות של הרכיב שלנו. אנו עושים זאת עם התבנית בשורה 11:

```
11:    For Each a In el.Attributes
```

התכונה Attributes

התכונה Attributes מחזירה רק את מה שהיינו מצפים: אוסף (Collection) של כל התכונות של הרכיב. בכל אופן, אוסף התכונות יש לו למעשה כניסה לכל תכונה אפשרית, כך שבשורה 14 אנו קוראים לשיטה המסוימת של האוסף כדי לראות אם כל תכונה נתונה היא אכן מסוימת עבור הרכיב הזה:

```
14:    If a.specified Then
```

אנו ממשיכים בשורה 15. ערכיהן של תכונות ב-HTML יכולים להיות תחת גרשיים או שלא. בכל מקרה, מנתח התחביר (Parser) חכם מספיק כדי להשיג את התכונה עבורנו ולהציג אותה כמחרוזת במודל האובייקטים. אנו לא יודעים ולא איכפת לנו אם זה במקור הופיע תחת גרשיים, כשאנו פולטים את התכונה אנו פשוט כותבים את זה נכון על ידי שימוש במרכאות כפולות ("").

הרווח הנקי הוא שאנו לא צריכים לדאוג ל"תיקון" המרכאות הכפולות במקור, אנו פשוט נותנים למנתח התחביר להשיג את הערך ואז אנו פולטים אותו החוצה בתצורה הנכונה. לאורך הדרך, אנו נחליף כל מרכאות כפולות בערך עם מרכאות בודדות, ונקיף את כל הערך במרכאות כפולות. כך שאם התחלנו עם

```
<p class = foo>
```

אנו נסיים עם

```
<p class="foo">
```

שורות 11-17 חוזרות באיטרציה על כל התכונות הקשורות עם הרכיב. למרבה העניין, תכונות הסגנון אינן באוסף התכונות, ולכן חובה לטפל בהן בנפרד. בשורה 22 אנו בודקים לראות אם יש לנו תכונת סגנון, ואם כן אנו מציגים את הערך באותיות קטנות, תוך תיקון המרכאות אם יש צורך.

בשורה 25 אנו בודקים אם לרכיב זה אין תגים-בנים (בעץ ההירכיה). אם אין, סיימנו, ואנו יכולים לפלוט את המחרוזת. אנו רואים זאת בשורה 26: אנו בודקים את המאפיין length של האוסף childNodes. אם הוא מחזיר 0, אזי האוסף ריק; אנו סוגרים את התגית בשורה 26 עם לוכסן עוקב, ובכך הופכים את התגית לריקה. (כפי שהסברתי קודם לכן, תגיות ריקות יש לסמן עם /> בסופן) ואז אנו יכולים להדפיס את המחרוזת בשורה 27 ולצאת מהשיגרה.

היה ונתקלנו ברכיב בעל רכיבי בנים, אנו ממשיכים בשורה 31. שוב אנו סוגרים את התגית הפותחת (הפעם ללא לוכסן, מכיון שזו תגית מכולה ואנו עומדים לתת לה תוכן). אנו מציגים את המחרוזת, אך הפעם אנו חייבים לעבור באיטרציה על כל הבנים של הרכיב הנוכחי.

שורה 36 מכינה את האיטרציה עם לולאת For, העוברת דרך כל בן (זכור, האוספים הם מבוססי-אפס, כך שאם האוסף מכיל חמישה רכיבים, הם ימוספרו מ- 0 עד 4):

```
36:   For i = 0 To el.childNodes.length - 1
```

בשורה 37 המשתנה המקומי n נקבע לבן הראשון, וככל שנתקדם באיטרציה, הוא ייקבע לכל בן עוקב:

```
37:   Set n = el.childNodes(I)
```

זכור ש-n הוגדר להיות אובייקט בלבד; הוא יחזיק בכל סוג של רכיב שנקבל מאוסף הבנים.

למעשה, בשורה 40 אנו חייבים לבחון את הטיפוס בפועל של הצומת שאנו מקבלים. הוא יהיה אחד משני טיפוסים אפשריים: רכיב או טקסט. אם ה-nodeType שלו הוא 1, זהו רכיב. הבה נדלג על מקרה זה לרגע אחד, ונבחן את המקרה השני (טקסט), הנראה החל משורה 44.

בשורה 46 אנו לוקחים בחשבון את הצגת צומת הטקסט. קבלת הערך כטקסט קלה; אנו משתמשים במאפיין nodeValue. הבעיה היא שהטקסט עשוי שלא להיות בדיוק מה שרצינו. ישנן שתי בעיות קרובות זו לזו.

ראשית, ל-Word יש תווים מיוחדים, כמו ציטוטים חכמים (Smart Quotes); אשר חייבים להיות מומרים לתגיות משלהם. בנוסף לכך, כאשר אנו קוראים את ה-HTML מן המסמך אל מודל האובייקטים של המסמך, ה-parser של IE5 משנה בנוחיות את הזהויות השקולות (Entity Equivalents) שלנו לטקסט (אלו הן השמות המקובלים לתווים מיוחדים, אשר פותחים בתו ה"אמפרסנד" &). למשל, אם ה-parser ראה < הוא מציג זאת כ- <, כמו כן, כאשר אנו רואים   ה-parser יציג זאת כתו של רווח. לרוע המזל, מכיוון שאנו כותבים זאת חזרה כמסמך, אנו זקוקים להציג זאת בחזרה לצורה המקובלת ב-HTML. עבודה זו מושגת על ידי השיטה ConvertTextNode כמוצג בתדפיס 2.5.

```
0:  'Perform the necessary transformations on a text node
1:  Private Function ConvertTextNode(s As String) As String
2:      'replace HTML special characters with their entity equivalents
3:      s = HTMLQuote(s)
4:
5:      'we want to do what follows _after_ the above,
        ➥since we really want these as tags
6:      'we'll replace Word "smart" quotes with special tags
7:      s = Replace(s, Chr(146), "<char type=""smartApos""/>")
8:      s = Replace(s, Chr(147), "<char type=""smartLQuote""/>")
9:      s = Replace(s, Chr(148), "<char type=""smartRQuote""/>")
10:
11:     'these are typographical shortcuts this publisher happens to use
12:     s = Replace(s, "[em]", "<char type=""emSpace""/>")
13:     s = Replace(s, "[md]", "<char type=""emDash""/>")
14:     s = Replace(s, "[lb]", "<char type=""bullet""/>")
15:
16:     ConvertTextNode = s
17: End Function
```

בשורה 3 אנו קוראים לשיטה HTMLQuote להמיר את כל תווי ה-HTML המומרים בחזרה לסימוני ה-HTML של היישויות השקולות. בשורות 7-9 אנו מטפלים בתווי ציטוט מיוחדים, ובשורות 12-14 אנו מטפלים בתווים מיוחדים שבשימוש ב-Macmillan Publishing (כמו קווים מפרידים ותבליטים).

רקורסיה

הבה נתייחס לשורה 42 של תדפיס 2.4. כאן אנו מתייחסים למקרה שבו הבן שאנו בוחנים (זכור, אנו חילצנו אותו מאוסף הבנים של הרכיב הנוכחי) הוא למעשה בן נוסף. מה אנו רוצים לעשות עם בן זה? למעשה, בדיוק את מה שעשינו עם הרכיב הנוכחי: ליצור מחרוזת עם התגית והתכונות, ואז למצוא את כל הבנים. זה בדיוק מה שהשיטה הנוכחית (OutputElement) מבצעת. צריך רק לקרוא לה, שוב, עם רכיב הבן:

```
42:    OutputElement n, indent & " "
```

כפי שאתה עשוי לדעת, מדובר ברקורסיה: כאשר פונקציה "קוראת" לעצמה. למעשה, הפונקציה אינה קוראת לעצמה. היא קוראת לעותק של עצמה, ושתי הפונקציות, בעוד שהן נקראות באותו שם ומשיגות את אותה מטרה, עובדות על קבוצות שונות לחלוטין של נתונים. השיטה OutputElement הראשונה עסוקה עם הרכיב הנוכחי, והשיטה OutputElement השנייה תעבוד על הבן. למעשה, הראשונה תחכה עד שהשנייה תסיים לפני שתמשיך.

כתיבת תגית הסגירה

הבה נחזור רגע אחורה לשורה 25. המקום שבו בדקנו האם ישנם רכיבים בנים.

אם אין כל רכיבים בנים, אז מבחן זה מחזיר אמת (אורך אוסף הבנים הוא 0) ואנו נכנסים לקוד בשורה 26. התגית נסגרת עם התגית המיוחדת של סגירה-עצמית />. לכן, אם הרכיב היה
 ולא היו בנים, אז אנו נפלוט אותו (בשורה 26) כ-
, שהוא התחביר הנכון עבור קבצי XML. זכור, ב-XML כל התגיות צריכות להיסגר. אם, מצד שני, יש לנו בנים, ה-if נכשל ובשורה 32 התגית נסגרת עם תגית סגירה רגילה.

מעבר דרך הקוד

כדי להפוך את כל זה למעט יותר מובן, אנו צריכים לעבור יחד על הפרטים. הדרך הטובה ביותר לעשות זאת היא באמצעות ה-debugger (מנפה השגיאות).

אתה תמצא את מודול המחלקה Word2XHTML על גבי התקליטור, או שתוכל להוריד אותו מאתר האינטרנט www.libertyassociates.com. טען מודול מחלקה זה בתוך הפרויקט FromScratch (מצוי גם כן על גבי התקליטור). כדי לגרום לזה לעבוד, תרצה גם להעתיק את control.asp לתיקיה בשרת ה-Web שלך, ואז ליצור את התיקיה הווירטואלית הנדרשת.



אם אינך בטוח לגבי אופן הכנת קובץ ASP על גבי שרת ה-Web או כיצד ליצור פרויקט חדש ב-VB, אנא פנה לנספח ד' לרשימת הקריאה המוצעת.

בחינת הקלט

כדי להבין על מה הקוד עובד, הבה נסתכל ב- 20 השורות הראשונות של קוד המקור Chap2.htm, כמוצג בתדפיס 2.6.

תדפיס 2.6

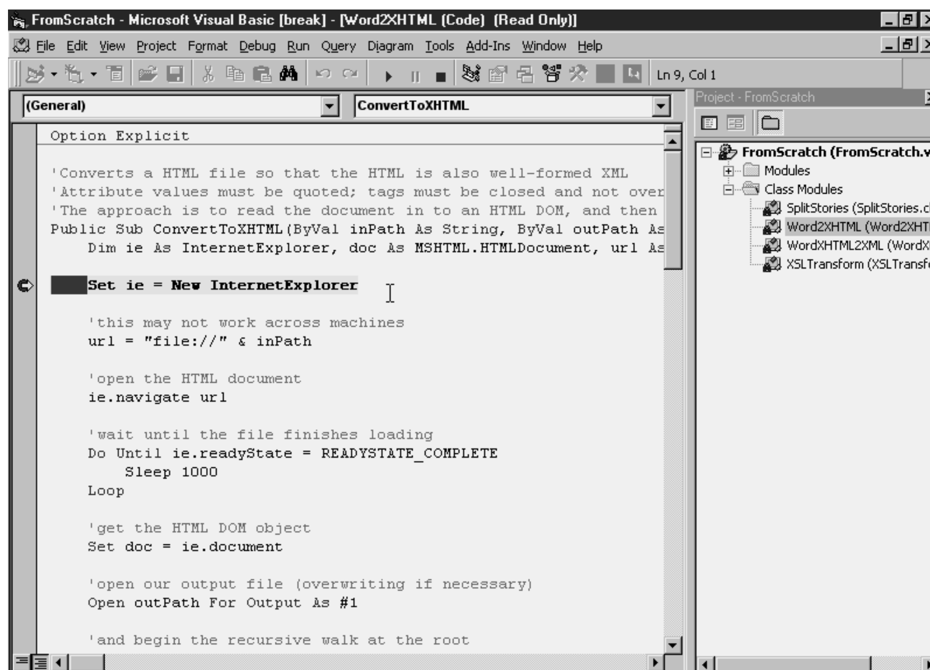
```
<html xmlns:o="urn:schemas-microsoft-com:office:office"
xmlns:w="urn:schemas-microsoft-com:office:word"
xmlns="http://www.w3.org/TR/REC-html40">

<head>
<meta http-equiv=Content-Type content="text/html; charset=windows-1252">
<meta name=ProgId content=Word.Document>
<meta name=Generator content="Microsoft Word 9">
<meta name=Originator content="Microsoft Word 9">
<link rel=File-List href="./Chap2_files/filelist.xml">
<title>(a) Chapter 12</title>
```

```
<!--[if gte mso 9]><xml>
 <o:DocumentProperties>
  <o:Author>Jesse Liberty</o:Author>
  <o:Template>mcpglobl.dot</o:Template>
  <o:LastAuthor>Jesse Liberty</o:LastAuthor>
  <o:Revision>2</o:Revision>
  <o:TotalTime>1</o:TotalTime>
  <o:LastPrinted>1999-07-12T23:11:00Z</o:LastPrinted>
  <o:Created>1999-10-27T11:16:00Z</o:Created>
```

חלק ניכר מזה נראה כמו קשקוש של Microsoft Office, אך אנו לא צריכים להבין מה זה עושה כדי לצפות בקוד במהלך פעולתו.

אני אשים נקודת עצירה (Break Point) על השורה הראשונה בעלת המשמעות ב-ConvertToXHTML, כמוצג בתרשים 2.7.



תרשים 2.7: break point ב-ConvertToXHTML

אנו נתחיל על ידי יצירת מופע של אובייקט ה-InternetExplorer, כפי שהוזכר קודם לכן. אנו יכולים לצעוד מהר דרך קוד זה עד שניכנס לשיטה OutputElement.

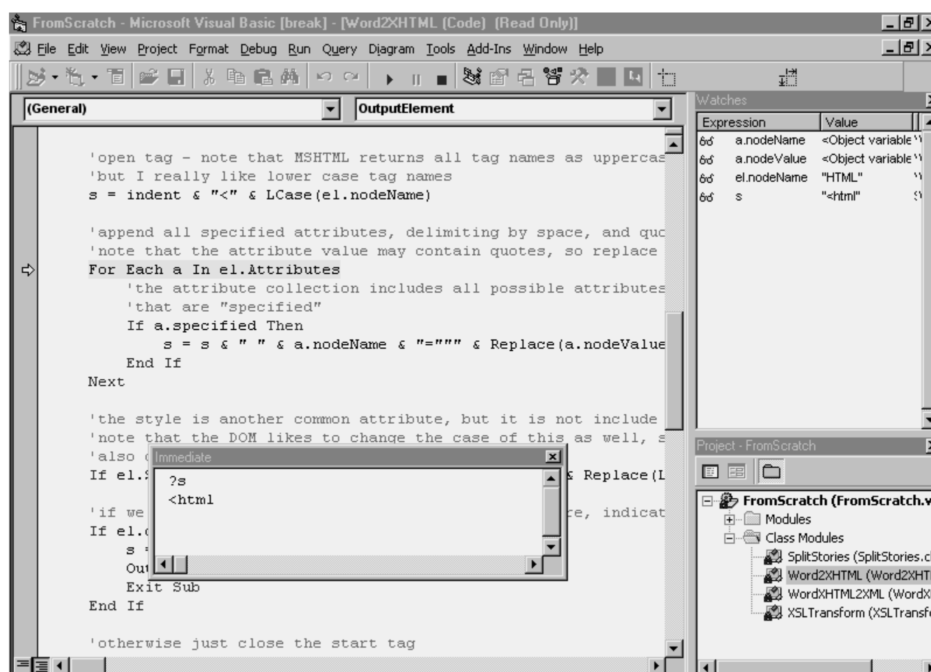


ב- Visual Basic, F9 ייצור נקודת עצירה ו-F8 יתקדם לתוך הקוד. Shift+F8 מדלג על קריאה לשיטה.

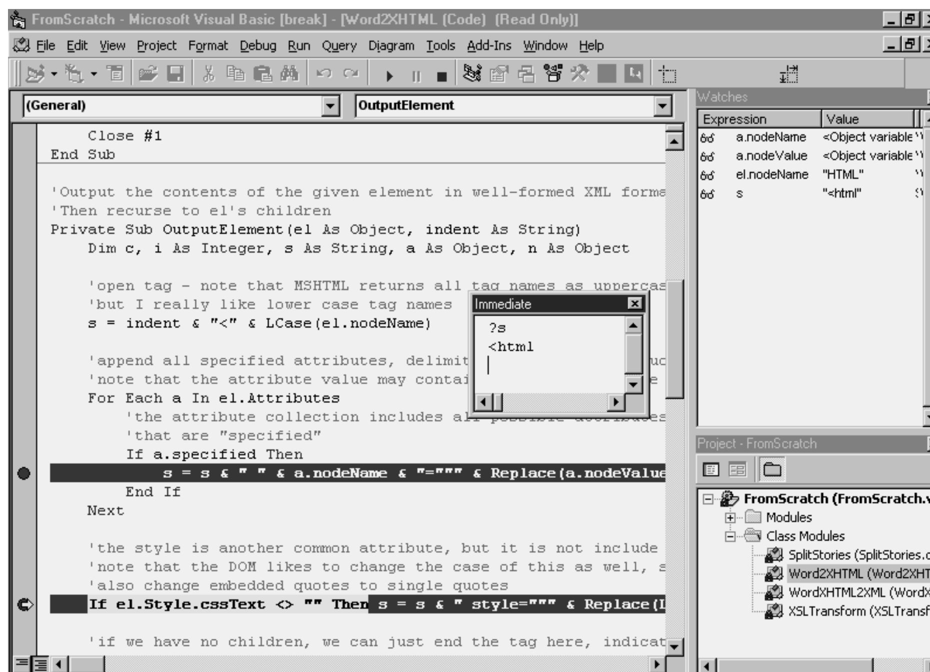
עצירה לאחר שיוך ה-nodeName למחרוזת שלנו, s חושפת שהנחותינו היו נכונות, כמוצג בתרשים 2.8. החלון המיידי, הנראה בתחתית המסך, מציג את הערכים הנוכחיים של s: <html, והחלון Watches, המוצג בחלק הימני העליון של המסך, מראה את הערך של el.nodeName, "HTML".

עד כה קיימת עקביות עם התיאור הקודם, הבה נמשיך. מכיוון שמרבית התכונות לא יסומנו, הדרך המהירה ביותר להגיע לשורה 11 של תדפיס 2.4 היא לשים נקודת עצירה עליה (F9). זכור לשים נקודת עצירה לאחר לולאת ה-For גם כן, במקרה שתגית ה-HTML הזו אינה מכילה תכונות. לך לנקודת העצירה הראשונה (F5).

אנו מוצאים עצמנו בשורה 22 של תדפיס 2.4, לאחר שדילגנו על כל התכונות הנראות בתרשים 2.9.



תרשים 2.8: בחינת המחרוזת



תרשים 2.9: שורה 22 מוצגת

כפי שניתן לראות, דילגנו על התכונות, והמחרוזת שלנו s מכילה רק <html. אם נביט חזרה בתדפיס 2.6, נוכל לראות שתגית ה-HTML נראית כך:

```
0:    <html xmlns:o="urn:schemas-microsoft-com:office:office"
1:       xmlns:w="urn:schemas-microsoft-com:office:word"
2:       xmlns="http://www.w3.org/TR/REC-html140">
```

ה-HTML DOM אינו מזהה אף אחת מתכונות אלו; והוא אינו צריך – אף אחת מהן אינה HTML. לכן, ה-HTML DOM אינו מזהה אף תגית מסומנת, ואנו ממשיכים לשורה הבאה של הקוד: בחיפוש אחר סגנונות.

בהמשך הקוד, אנו מדלגים על הצהרת ה-if בשורה 22 של תדפיס 2.4

```
22:   If el.style.cssText <> " " Then s = s & " style=" " " &
      ➥Replace(Lcase(el.Style.cssText), " " " ", " ' ") & " " " "
```

מכיוון שאין לנו כלל תגיות סגנון ברכיב זה, ואנו מדלגים על הצהרת ה-if בשורה 25

```
25:   If el.childNodes.length = 0 Then
```

בגלל ש-el.childNodes.length אינו אפס, וגם לא נצפה שיהיה. זכור ש-el במקרה זה הוא צומת השורש (HTML), childNodes הוא האוסף של הבנים של HTML, ו-length הוא המונה של הצמתים באוסף זה.

אנו ממשיכים עם שורות 36-42 של תדפיס 2.4.

```
36:     For i = 0 To el.childNodes.length - 1
37:       Set n = el.childNodes(i)
38:
39:       'In HTML, a node will either be an element or a text node
40:       If n.nodeType = 1 Then
41:         'an element - recurse
42:         OutputElement n, indent & "  "
```

מסתבר ש-n (הבן הראשון של <HTML>) הוא רכיב, כך שנקרא ל-OutputElement עם n (הבן שזה עתה קיבלנו) ועם מחרוזת ההזחה שלנו, אך עם הוספת זוג רווחים למחרוזת ההזחה.

אם תלחץ על F8 משורה 42, תחזור לכאורה להתחלה של OutputElement. אבל לא חזרת להתחלה, פשוט ירדת רמה ברקורסיה. כלומר, הפונקציה שבה עבדת הושהתה. אם נחזור לאותה פונקציה מוקדמת el הוא <HTML>. כאן, בכל אופן, el לא יהיה HTML, הוא יהיה <head>, הבן הראשון של HTML.

שוב, לא נמצאו תכונות או סגנונות, ושוב ישנם בנים. למעשה, הפעם ישנם שבעה בנים. שוב, אנו משייכים ל-n את הבן הראשון, ושוב אנו יורדים רמה ברקורסיה לתוך OutputElement. הפעם el הוא הבן הראשון של <head>: <title>. רכיב זה, <title>, חסר בנים, כך שאנו נכנסים להצהרת ה-if בשורה 25:

```
25:   If el.childNodes.length = 0 Then
26:       s = s & " />"
27:       out s
28:       Exit Sub
29:   End If
```

אנו סוגרים את התגית ויוצאים מהשיגרה. לאן אנו יוצאים? אנו "קופצים החוצה" רמה אחת מעלה ברקורסיה לשיטה OutputElement, שקראה לנו: זו האחת שבה <head> הוא el. זה מחזיר אותנו לשורה 37, שורת האיטרציה על הבנים. אנו עברנו באיטרציה על הבן הראשון של <head>, שהיה <title>. מכיוון של-<title> אין בנים, אנו מוכנים להמשיך באיטרציה לבן השני של <head>: <meta>.

שים לב שאנו עדיין בבן הראשון של <HTML>, שהיה <head>, אך עברנו לבן השני של <head>, <meta>. אם נחזור למסמך המקור, הנראה בתדפיס 2.6, אנו כרגע בשורה 5:

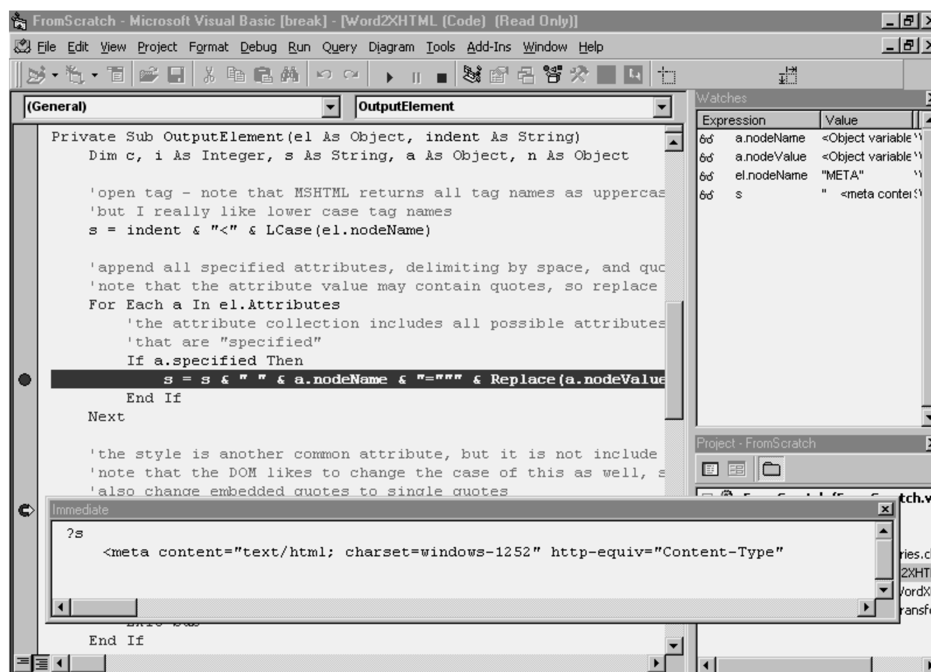
```
5:  <meta http-equiv=Content-Type content="text/html; charset=windows-1252">
```

ה-DOM קבע ש- <title> (שורה 10) היה הבן הראשון, ו- <meta> (שורה 5) הוא הבן השני. ראה זאת כאזהרה: הסדר אינו מובטח.

אנו ממשיכים להתקדם בקוד, והפעם ה-parser מוצא שסומנה תכונה, ומפסיק בשורות 14-16 כדי לאסוף אותה:

```
14:   If a.specified Then
15:        s = s & " " & a.nodeName & "=""" & Replace(a.nodeValue, """", "'") & """"
16:   End If
```

אנו רצים בלולאה דרך כל התכונות, ומעצבים אותן כהלכה כמודגם בחלון המיידי בתחתית תרשים 2.10.



תרשים 2.10: לאחר קליטת התכונות

ל-<meta> אין כלל בנים, כך שאנו ממשיכים לבן הבא מבין שבעת הבנים של <head>, שהוא גם כן תגית <meta>. למעשה, ישנן מספר תגיות <meta> בהן נתַקל לפני שנגיע לתגית <link>, המייצגת את שורה 9 במסמך המקור שלנו:

```
9:     <link rel =File -List href="./WCFS%20Chapter%201_files/filelist.xml">
```

בשלב הבא אנו אוספים בן <style> של <head>, משורה 56 בקוד המקור, כמוצג בתדפיס 2.7.

תדפיס 2.7 מובאה מתוך המקור

```
56:  <style>
57:  <!--
58:   /* Font Definitions */
59:  @font-face
60:       {font-family:Courier;
```

```
61:       panose-1:0 0 0 0 0 0 0 0 0 0;
62:       mso-font-charset:0;
63:       mso-generic-font-family:modern;
64:       mso-font-format:other;
65:       mso-font-pitch:fixed;
66:       mso-font-signature:3 0 0 0 1 0;}
67:   /* Style Definitions */
68:  p.MsoNormal, li.MsoNormal, div.MsoNormal
69:       {mso-style-parent:"";
70:       margin:0in;
71:       margin-bottom:.0001pt;
72:       line-height:12.0pt;
73:       mso-pagination:widow-orphan;
74:       font-size:12.0pt;
75:       mso-bidi-font-size:10.0pt;
76:       font-family:Courier;
77:       mso-fareast-font-family:"Times New Roman";
78:          mso-bidi-font-family:"Times New Roman";}
```

מכיוון שכל הסגנונות הללו מצויים בתוך הערת HTML, איננו אוספים כל תכונה או בנים עבור <style> ובמהרה ממשיכים בחזרה באיטרציה על ילדיו של <head>. בכל אופן, זהו הבן האחרון של <head>, כך שהפונקציה הרצה באיטרציה על בניו של <head> מסתיימת.

לאחר שעברנו דרך כל בניו של <head>, לולאת ה-For שרצה משורות 35-48 של תדפיס 2.4 מסתיימת, ואנו ממשיכים הלאה לשורה 51 :

```
51:    Out indent & "</" & LCase (el.nodeName) & ">"
```

זה שולח לקובץ הפלט שלנו סדרה של רווחים (מחרוזת ההזחה), כמו גם תגית סגירה, בהתבסס על שם התגית.

לאחר שסגרנו את התגית, השיטה שלנו חוזרת. לאן? חזרה מעלה ברמה אחת לפונקציה הרצה באיטרציה על שני בניו של <HTML>. אנו, בקצרה, מוכנים עתה לבן השני של <HTML> : <body>, המצוי בשורה 1,538 של מסמך המקור שלנו :

```
1538: <body lang=EN-US style='tab-interval:.5in'>
```

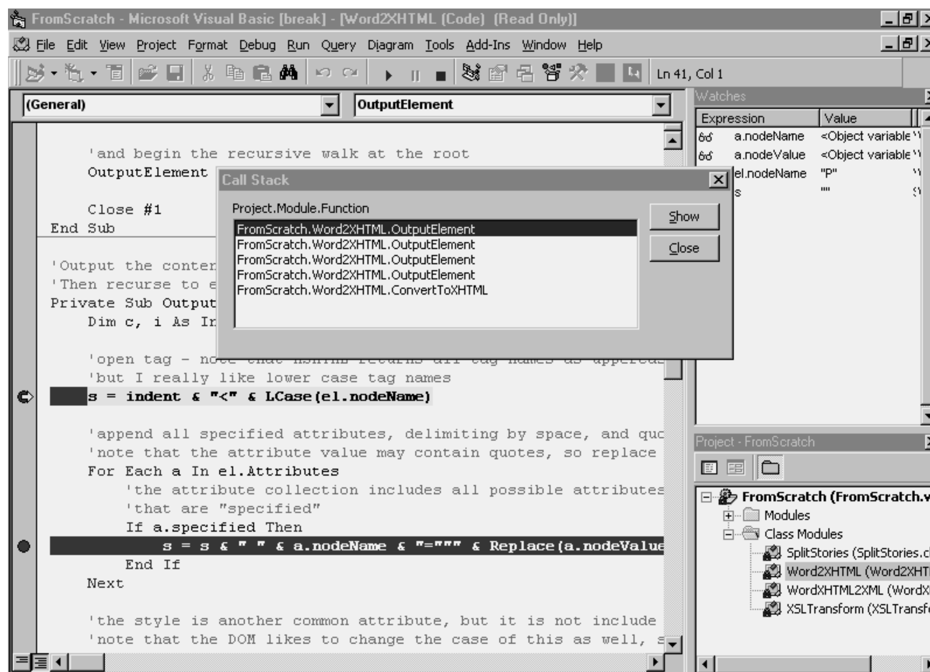
בחינת התוצאות

התגית <body> מייצגת את הטקסט שיוצג במסמך HTML. התגית <head>, בוודאי תזכור, הכילה מטה-מידע ; בתגית ה-<body> נעשית עיקר העבודה.

בתגית זו אנו רואים לראשונה את השימוש בסגנונות. השינוי היחידי שנעשה לקוד המקורי הוא שאנו כופים על תכונת הסגנון להיות מוקפת במרכאות כפולות.

מהתגית <body> אנו מקבלים את הבנים, ואנחנו מוצאים תגית <div>; מהתגית <div>, הבנים מובילים אותנו לתגית <p>.

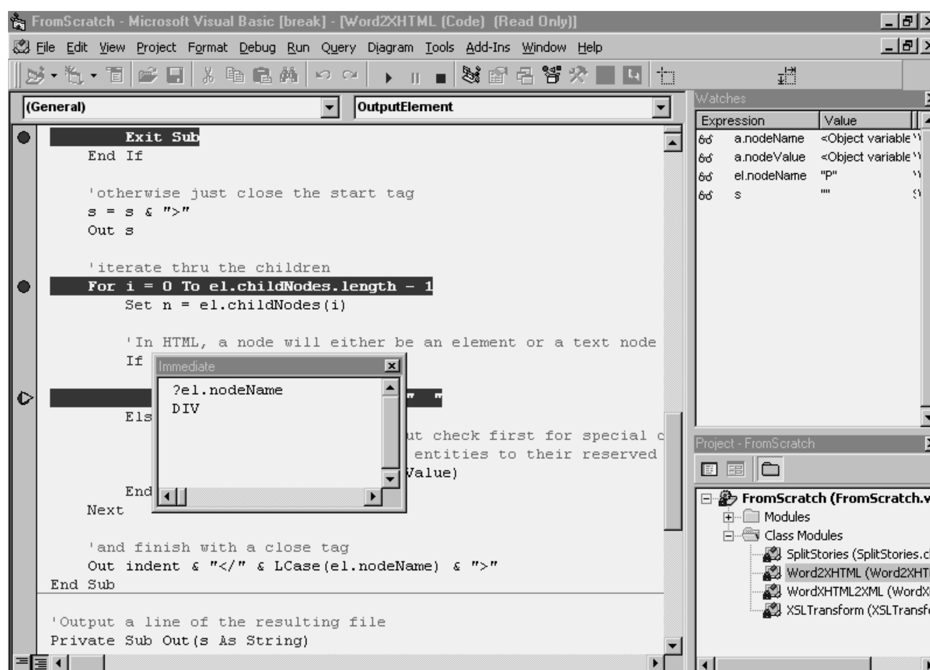
בזמן בחינת התגית <p>, לחץ על Ctrl+L כדי להעלות את חלון מחסנית הקריאה, כמוצג בתרשים 2.11.



תרשים 2.11: מחסנית הקריאה (Call Stack)

כאן אנו רואים את הרקורסיה בפעולה. קריאת הפונקציה היא בסדר הפוך (הכי מאוחרת תהיה הראשונה). לכן, ConvertTOXHTML קרא ל-OutputElement, שקרא ל-OutputElement, שקרא ל-OutputElement, שקרא בתורה ל-OutputElement.

אם נסתכל בחלון Watch, נוכל לראות שה-el.nodeName הוא p. על ידי לחיצה על קריאה אחת לפונקציה ברשימה, נוכל לראות שה-el.nodeName של הקריאה הקודמת לפונקציה הוא Div, כמוצג בתרשים 2.12.



תרשים 2.12: החלון Watch המראה את שם הקריאה לפונקציה

לחיצה על רמת רקורסיה אחת נוספת מעלה, תראה כי el.nodeName הוא Body, והרמה הבאה תראה ש-el.nodeName הוא HTML, כפי שהיינו מצפים. כך שאנו נמצאים בבן של Div, שהוא בן של Body, שבתורו הוא בן של HTML.

תן לאצבעות ללכת...

התוכנית ממשיכה באופן זה; לקיחת כל תגית וכל בניה, ברקורסיה לתוך עצמה עד שכל העץ נסרק, נקרא ונוצר מחדש כמסמך הפלט שלנו.

אין כל פלט למשתמש לאורך הדרך (מושאר כתרגיל לקורא), כך שאתה חייב להתאזר בסבלנות עד שהתוכנית מדווחת שהיא סיימה את ההמרה. זה יכול לקחת עד מספר דקות, תלוי בגודל קובץ הקלט.

כאשר התוכנית תסיים, אתה יכול לבחון את קובץ ה-XHTML המתקבל.

תדפיס 2.8 מדגיש את 12 השורות הראשונות.

תדפיס 2.8 קובץ ה-XHTML

```
0:  <html>
1:    <head>
2:      <title />
3:      <meta content="text/html; charset=windows-1252"
        ➥ http-equiv="Content-Type" />
4:      <meta content="Word.Document" name="ProgId" />
5:      <meta content="Microsoft Word 9" name="Generator" />
6:      <meta content="Microsoft Word 9" name="Originator" />
7:      <link href="./Chap2_files/filelist.xml" rel="File-List" />
8:      <style />
9:    </head>
10:   <body lang="EN-US" style="tab-interval: .5in">
11:     <div class="Section1">
12:         <p class="FT">
```

המבנה של התגיות מיוצג גרפית על ידי הזחה. התגית הראשונה (השורש) היא <html>, התגיות <head> ו-<body> מוזחות רמה אחת פנימה. תחת <body> מופיע <div> ותחת <div> ישנו <p>, בדיוק כפי שראינו במעקב אחר הקוד.

שים לב, שכעת כל התכונות מופיעות במרכאות כפולות, וכל התגיות מקוננות כהלכה. אתה יכול לראות תגיות בודדות, הסוגרות את עצמן בשורות 2 ו- 8.

הצעדים הבאים

עד כה, שמרנו קובץ Word כקובץ HTML. תוכנת Word ייצאה את הקובץ כ-HTML שאינו בנוי היטב. כדי להפוך את הקובץ לקובץ XHTML בנוי היטב היה עלינו להעבירו דרך תוכנית אשר תמיר את הקוד לקוד XHTML בנוי היטב. כלומר, תדאג לכך, שהקוד עומד בדרישות הקשיחות של XML. את המעבר עשינו בעזרת תוכנית אשר יצרנו ב-VB, אשר רצה על התגיות עצמן, ובעזרת ה-DOM של HTML, יצרנו מחדש קובץ HTML אשר עומד בדרישות של XML. במילים אחרות - יצרנו קובץ XHTML.

הצעד הבא יהיה להפוך את קובץ ה-XHTML לקובץ XML.

פרק 3
DTD, מתן חוקיות למסמך

בפרק זה:

- מ-XHTML ל-XML
- הבטחת חוקיות
- מטרות המעבר
- יצירת ה-DTD
- הצעדים הבאים

תוצאת העבודה של פרק 2 היא שעתה יש לנו מסמך XHTML. למעשה, יש לנו שלושה קבצים. הקובץ הראשון הוא קובץ Word המקורי. השני, הוא קובץ html שיצר Word כששמרנו את המסמך כדף אינטרנט. הקובץ השלישי הוא קובץ ה-XHTML שיצרנו בפרק הקודם. כעת נותר לנו להפוך את תוכן קובץ XHTML לתצורת ה-canoncial storage שלנו. את המעבר נעשה לאורך הפרקים הבאים עד פרק 5.

מ-XHTML ל-XML

ודאי שאלת כבר בסוף פרק קודם, מדוע עלינו להמיר את מסמך ה-XHTML ל-XML, הרי, כפי שהוסבר קודם, מסמך זה הוא גם HTML חוקי וגם XML חוקי. כלומר, הוא עומד בשני המפרטים. ככזה, הוא כבר מסמך XML. למה, אם כן, עלינו להמיר אותו ל-XML? למה לא לשמור אותו ולאחסן אותו כמו שהוא?

כפי שהוא, המסמך מכיל בעיקרו תגיות תצוגה של HTML. זה מתאים למסמך HTML, ואם התכוונו להשתמש במסמך זה באינטרנט בלבד, היה בסדר גמור לאחסן אותו כפי שהוא. אבל, מטרתנו היא להשיג גמישות גדולה בהרבה. ייתכן ונרצה לפרסם מסמך זה בתצורות שונות, בפלטים שונים או במדיות אחרות. אחסון המסמך ב-HTML הופך את המשימה לקשה יותר, אחסונו ב-XML הופך אותה לקלה יותר. ייתכן ונרצה לבחור אלו מן הקטעים להציג וכיצד לעשות זאת, בהתבסס על מידע דינמי (כגון פרופיל המנוי).

שוב, XML יאפשר זאת בקלות. הדרישה לכך שנהיה מסוגלים לפרסם את המסמך לא רק ברשת האינטרנט, אלא גם בהדפסה, דואר אלקטרוני, ובהתקנים אחרים כגון מנהלי מידע אישיים (PIM, Personal Information Manager), מחשבי כף-יד כגון PalmPilot, PocketPC או Cassiopeia) וכן הלאה, מהווה אתגר ל-HTML, אך קלה ביותר עם יכולותיה של XML. שכן, לאחר הפיכת המסמך למסמך XML, נחזיק בידינו תגיות ייחודיות בעלות משמעות תכנית, אשר תכלנה את המידע הנדרש, ללא כל קשר לאופן הצגתו. לאחר מכן, כשנרצה להציגו במדיות שונות, נוכל להעביר את מסמך ה-XML דרך מפרשים (Parsers) מתאימים, אשר ייצרו את הפלטים על פי דרישותינו. לשם הבהרה: הדפדפן - הוא Parser בפני עצמו. כאשר אנו מעבירים מסמכי XHTML דרכו, הוא מפענח את הקוד ומציג אותו למשתמש באופן ויזואלי. הוא משמש כמפענח למטרות הצגת הנתונים ב-WWW.

מסמך ה-XHTML מכיל תגיות אשר מייצגות סגנונות הצגה ויזואלית. אנחנו מעוניינים במסמך XML, המורכב מתגיות בעלות משמעות תכנית, לכן, עלינו לבטל את הסימון הייחודי לתצוגה מהפרטים הסמנטיים והמבניים. אנו נשמור רק את התוכן ואת נתוני העל המתארים את התוכן ואת מבנהו.

הפיכת המבנה למפורש

מניע אחד חשוב להפיכת מסמך XHTML שלנו ל-canoncial form הוא שמסמך ה-XHTML (כמו מסמכי Word ו-HTML, שהם אבותיו הקודמים) אינו מספק מידע מפורש (Explicit) אודות מבנה המסמך. כלומר, מסמך XHTML אינו מראה כותרות ברמה-C הסוגרות על כותרות ברמה-D, הן פשוט ברצף, אחת אחרי השנייה. העובדה שכותרות ברמה-D מוכלות בכותרות ברמה-C היא מרומזת (Implicit) – זה מובן על ידי העורך, אך זה לא מפורש במבנה המסמך. בכל אופן, מבנה מרומז זה הוא מהותי למשמעות המסמך, ואנו רוצים להפוך אותו למפורש במסמך ה-canoncial form שלנו.

אנו כופים מבנה זה על המסמך המודפס, על ידי עמידה במוסכמות של עריכת הדפוס. העורך יודע, על פי הסכם הדדי, שקטעים מסומנים בכותרות ברמה-C, וקטעי משנה בכותרות ברמה-D. עם ידע זה, מסוגל העורך ליצור טבלה של תוכן העניינים המשקפת את המבנה הפנימי. היא אינה מפורשת ב-XHTML, היא מרומזת; נבנית על ידי אדם היודע את החוקים.

היינו רוצים לקדד חוקים אלו במסמך ה-XML שלנו, ולהפוך את המבנה למפורש. הדבר יאפשר לנו להפוך את התהליכים לאוטומטיים, כשכרגע הם דורשים התערבות אנושית.

הבה נהיה ברורים: ה-canoncial form היא canoncial רק לנו. מסמך ה-XHTML הוא כבר XML חוקי. אין שום דבר קסום או מיוחד ב-canoncial form ; הצורה הזאת ייחודית ליישום זה בלבד. ליישומים אחרים יהיו canoncial form אחרים משלהם.

אנו נלכוד את הדקדוק של ה-canoncial form שלנו בהגדרת טיפוס המסמך, DTD (Document Type Definition).

דקדוק (Grammer) - התחביר והמבנה של מסמך.

הגדרת טיפוס מסמך, Document Type Definition (DTD) – מסמך המגדיר את הדקדוק החוקי של מסמך XML.

על ידי בחינת והבנת ה-DTD הזה, אתה תראה שיצירת DTD משלך לתצורת canoncial storage (אחסון) משלך היא מטלה קלה למדי.

הבטחת חוקיותם של מסמכי XML

יש מספר דרכים בהן DTD (Document Type Definition) מגדיר את מבנה מסמך XML, ולכן מגדיר גם את הדקדוק של ה-canoncial form שלנו. למשל, ה-DTD מגדיר את שמות התגיות בהן נעשה שימוש במסמך. אם שם תגית אינו מופיע ב-DTD, אז אסור לו להופיע במסמך XML שלנו. אם תגית לא ידועה כזו אכן מופיעה, אז אנו אומרים שמסמך XML "**אינו חוקי**". כלומר, הוא הפר את האילוצים המוכתבים על ידי ה-DTD שלו.

ייתכן שהתבלבלת בין שני המושגים "חוקי" ו-"בנוי היטב". ההבדל בין שניהם הוא מהותי, וחשוב להבנה עוד בשלבים המוקדמים של עבודתך:

מסמך XML חוקי – הוא מסמך אשר עומד בדרישות והאילוצים המוכתבים על ידי ה-DTD שלו. כמובן, שאם במסמך תגיות או ערכים אשר אינם רשומים ב-DTD שלו, מסמך זה אינו חוקי.

מסמך XML בנוי היטב (Well-Formed) – מסמך XML בנוי היטב אם הוא עומד במפרט XML. בין דרישות אלו נמצא שאסור שתגיות מקוננות יחפפו, שמות תגיות חייבים להיות תלויי רישיות, כל התגיות חייבות להיסגר, והתכונות חייבות להיות תחת מרכאות. מסמך יכול להיות בנוי היטב אפילו אם אינו חוקי.

עדכון ה-DTD

ה-DTD מגדיר אילו תגיות יכולות להופיע במסמך ה-XML. אם אנו רוצים להוסיף תגית חדשה, אנו חייבים לעדכן את ה-DTD שלנו. זה יהפוך את התגית החדשה לחוקית, והוספתה למסמך תשאיר אותו חוקי.

ה-DTD מגדיר גם את היחסים המבניים בין התגיות – כלומר, אילו תגיות מכילות תגיות אחרות, וכן אלו תגיות הן הכרחיות או אופציונליות. ניתן אף להגדיר את הסדר בו תגיות יופיעו. ה-DTD מתאר גם את התכונות ההכרחיות או האפשריות לכל רכיב.

שוב, אם אילוצים אלו מופרים, אז המסמך כבר לא יהיה חוקי (אפילו שהוא עשוי להיות בנוי היטב!)

מסמך אינו דורש DTD מפורש, אך אם מסופק כזה, מנתח התחביר (Parser) יהיה מסוגל לדווח לא רק האם ה-XML בנוי היטב (כלומר, הוא עומד בסטנדרט XML), אלא גם האם המסמך חוקי, כלומר, האם הוא עומד בדרישות של ה-DTD.



ישנם מספר מנתחי תחביר (parsers) הזמינים באינטרנט ומ-W3C. מנתח התחביר בו נשתמש בספר זה הוא MSXML. MSXML הוא מנתח תחביר XML של Microsoft, והוא כלול כאובייקט ActiveX עם IE5 בשם MSXML.DLL. אם תתקין את IE5, תמצא את MSXML בתיקיה WinNT\system32 (במערכת Windows NT/2000).

מנתח תחביר המסוגל לוודא חוקיות של מסמך מול ה-DTD שלו נקרא **מנתח תחביר מוודא חוקיות** (Validating Parser). מנתח תחביר המתעלם מה-DTD נקרא **מנתח תחביר שאינו מוודא חוקיות** (Non-Validating Parser). MSXML (ולכן גם IE5) הוא מנתח תחביר המוודא חוקיות.

מנתח תחביר מוודא חוקיות

מנתחי תחביר המוודאים חוקיות מורכבים הרבה יותר מאשר אלה שאינם מוודאים חוקיות, אך הם מספקים כלים נוספים לשימוש בעת יצירת המסמכים. מנתח תחביר המוודא חוקיות יעיר על שגיאה אם הוא נתקל בהפרה כלשהי במסמך אשר נוגדת את האילוצים של ה-DTD.

זוהי תכונה רצויה במנתח תחביר, מפני שהיא הופכת את מתן החוקיות, איתור השגיאות ותיקון המסמך למהירים וקלים יותר. זהו אותו הטיעון לטובת השימוש בשפות עם הגדרות קשיחות (Strongly Typed Languages), כדוגמת C++, או הפעלת האפשרות Option Explicit ב-Visual Basic, או שימוש ב-Declared Referential Integrity ב-SQL. המשותף לכל אלו הוא הגיוס של התוכנה לעזרת ההגנה על שלמות קוד המקור; כל אלו עוזרים לך להימנע משגיאות עוד **לפני** שאתה שולח את המוצר ללקוח שלך.



ראוי לציין שאפילו אם אינך מספק DTD מפורש, בהחלט יש בידך אחד מרומז, המאוחסן אולי רק במוחך. ככלות הכל, אתה יודע באלו תגיות תשתמש ומה משמעותן; DTD פשוט כותב את החוקים האלו כך שהדפדפן יוכל לתת משנה תוקף למסמך שלך על סמך האילוצים שלך. הדפדפן בודק אותך, שלא טעית באילוצים שקבעת בעצמך.

DTD יכול להיות בעל ערך גם כאשר יותר מאדם אחד (או קבוצה) עובדים על סט של מסמכים או כלים. ה-DTD אומר לכל קבוצה אלו תגיות הן חוקיות וכיצד הן צריכות להיות מקוננות. אתה יכול לראות כיצד זה יהיה בעל ערך עליון לשמירת האחידות בעבודת צוות. אם תגית חדשה תתווסף, הוא תושם תחילה ב-DTD, כך שכולם ידעו את התחביר שלה ואת משמעותה.

DTD יכול לשמש גם ככלי עיצוב טוב. על ידי בניית ה-DTD לפני התחלת הקידוד, ניתן לעצב את מבנה מסמך ה-XML, במקום לאפשר לו להתהוות בצורה אקראית עם התכנות.

רמת בניית ה-DTD **לפני** כתיבת קוד כלשהו היא גם עניין של סגנון אישי וגם פונקציה של גודל הפרויקט. בפרויקט גדול, בייחוד כזה המערב מפתחים רבים, DTD מפורש הוא בעל חשיבות עליונה.

כפי שהחדירו לנו בקורסי המבוא לתכנות: התכנון מראש הוא אחד השלבים החשובים ביותר בתכנות אפליקציה.

על אותו משקל: בניית DTD נכון ומפורט ככל שניתן, היא צעד חשוב בתכנון מראש של אפליקציית XML.

מטרות המעבר

ה-DTD מגדיר מסמך BiblioTech חוקי. כדי ליצור DTD אנו חייבים לדעת אלו תגיות אנו רוצים לאפשר ומה צריך להיות המבנה הכולל של המסמך. ה-DTD מציין חוקים אלו.

זוהי נקודה מרכזית. באופן דומה, זו תהיה שגיאה לחשוב שחוקת ארה"ב יצרה את החופש שלנו. ההיפך הוא הנכון - כותבי החוקה, ראו את מטרתם כחופש. על פי כך הם כתבו את החוקה בהתאם למטרותיהם.
במקביל, עד שלא נדע אילו חוקים ברצוננו לאכוף, יכולתנו לכתוב DTD אינה מספקת. ברגע שאנו יודעים מה אנו רוצים להשיג, הקושי היחידי בכתיבת DTD הוא רק בהקפדה על תחביר נכון.

האילוצים שברצוננו לאכוף

בבחינת מסמך ה-XHTML, אנו מוצאים שהוא יותר מדי ממוקד בסוגיות של תצוגה והצגה. אנו רוצים להבטיח ש-canoncial form שלנו היא מידע ומבנה ממוקדים. אנו נבצע שישה סוגים של שינויים כדי לעבור מ-XHTML ל-XML ה-canoncial form שלנו:

1. הסרת כל ה-HTML שאיננו צריכים, כמו התגיות <meta> ו-<style>, או קוד HTML כזה שאינו מיושם, כמו התגית <html>.
2. סימון כל שורות הקוד כך שנוכל לנהל אותן בנפרד.
3. אחסון רווחים וטאבים ביתר יעילות.
4. סימון כל ה-"**מונח טכני**", "כיצד מבטאים זאת?", הערות וסרגלי צד (Sidebars) אחרים כך שנוכל לנהל אותם בנפרד.
5. שבירת קטעים, ובניית ההיררכיה המקוננת, תוך זיהוי כל קטע באופן ייחודי.
6. מעבר על כל חלקי ה-HTML בעלי המשמעות.

הבה נבחן מטרות אלו ביתר פירוט.

הסרת HTML שאיננו צריכים

ישנן תגיות HTML רבות שפשוט איננו צריכים במסמך ה-XHTML. הבה נסתכל ב- 29 השורות הראשונות של קובץ ה-XHTML, בתדפיס 3.1.

תדפיס 3.1

```
<html>
   <head>
     <title />
     <meta content="text/html; charset=windows-1252"
       ➥http-equiv="Content-Type" />
     <meta content="Word.Document" name="ProgId" />
     <meta content="Microsoft Word 9" name="Generator" />
     <meta content="Microsoft Word 9" name="Originator" />
     <link href="./Chap3_files/filelist.xml" rel="File-List" />
     <style />
   </head>
   <body lang="EN-US" style="tab-interval: .5in">
  <div class="Section1">
     <p class="HA">
(a)3
     </p>
     <p class="HB">
(b)Proof of concept
     </p>
     <p class="FT">
Enough theory! Before we go any further in thinking through how we'll implement
➥EmployeeNet, we need to take a look at the implementation technology and get
➥something working.
     </p>
     <p class="FT">
 While I believe in analysis and design, my number one rule of programming is this:
     </p>
     <p class="FT">
       <b style="mso-bidi-font-weight: normal">
       <i style="mso-bidi-font-style: normal">
Get something working right away, and keep it working until it&apos;s done
       </i>
```

ניתוח

אנו רואים שישנן מספר תגיות שהן ייחודיות ל-HTML. למשל, אנו יכולים להיפטר מן התגית <meta> בשורות 5,4,3 ו- 6 שכן הן חסרות משמעות במסמך ה-XML שלנו, ואינן מספקות מידע אודות תוכן המסמך.

כדי להיות ברור יותר: התגית <meta> נדרשת במסמך HTML, אך אינה נדרשת במסמך ה-XML שלנו וכמובן שלא תופיע ב-DTD שלנו. לכן, חייבים להורידה. באופן דומה, נזרוק את התגית <head>, שכן היא חסרת משמעות במסמך ה-XML שלנו.

אם וכאשר נפרסם מסמך זה על גבי רשת האינטרנט, אנו ניצור מסמך HTML חדש, ותגית <head> חדשה. שום נתון מתוך הנתונים הנמצאים כרגע בתגיות <meta> או <head> של המסמך לא יידרש, כך שאין כל פסול בהחלטה שלא לאחסן אותן.

עוד בנושא זה, אנו נהפוך את התגיות <p> לתגיות <div>. אני אציין את התגיות האחרות שאנו לא צריכים ככל שנתקדם, אך הפילוסופיה היא להתמקד בתוכן ובמבנה, ולהתעלם ממידע אודות מאפייני תצוגת HTML ייחודיים.

סימון שורות קוד

השינוי השני שאנו רוצים להשיג הוא ביצירת בלוקי קוד ובלוקים של הערות. אם תבחן את הגירסה המודפסת של ספר זה, כמוצג בתרשים 3.1, תראה שהעין האנושית יוצרת באופן טבעי בלוקי קוד או בלוקי הערות בהתבסס על גופנים, הצללה, רווח ומוסכמות הדפסה אחרות. כרגע אין שום דבר מהותי במסמך ה-XHTML שלנו כדי ליצור את הבלוקים האלו. ה-XHTML פשוט מספק את הסגנונות והרמזים למו״ל, אך המבנה הוא לחלוטין מרומז. אנו נרצה להפוך אותו למפורש. כדי לעשות זאת, אנו נצרף בלוקים עוקבים של קוד בין תגיות בעלות משמעות דומה. נעשה את אותו הדבר עם הערות, "**מונח טכני**", ושאר הבלוקים המייצגים הינתקות מהזרימה הנורמלית של הספר.

אם נבחן את ה-XHTML, נמצא שהקוד מסומן על ידי מזהה תדפיס, ושכל שורת קוד נקבעת לסגנון C2, פרט לשורה האחרונה הנקבעת ל-CX. שורות בודדות של קוד נקבעות לסגנון C1. ראה לדוגמה תדפיס 3.2.

תדפיס 3.2

```
Function GetTheData() As Recordset
    Dim rs As ADODB.Recordset
    Set rs = New Recordset
    Call rs.Open("select * from publishers", "dsn=pubs",  UID=sa; PWD=;")
    Set GetTheData = rs
End Function
```

כאשר Word הופך זאת ל-HTML, סימני סגנונות אלו נשמרים כמוצג בתדפיס 3.3

תדפיס 3.3

```
<p class=C2>1: Function GetTheData() As Recordset</p>

<p class=C2>2:<span style="mso-spacerun: yes">     </span>Dim rs As
ADODB.Recordset</p>

<p class=C2>3:<span style="mso-spacerun: yes">     </span>Set rs = New
Recordset</p>

<p class=C2>4:<span style="mso-spacerun: yes">     </span>Call
rs.Open(&quot;select * from publishers&quot;, &quot;dsn=pubs&quot;,<span
style="mso-spacerun: yes">  </span>UID=sa; PWD=;&quot;)</p>

<p class=C2>5:<span style="mso-spacerun: yes">
➥ </span>Set GetTheData = rs</p>

<p class=CX>6: End Function</p>
```

Listing 3.4

```
Option Explicit
Option Compare Text

Private Sub Publishers_ProcessTag_
(ByVal TagName As String, TagContents As String,_
 SendTags As Boolean)

    Select Case TagName

    Case "WC@Publishers"
         Dim rs As Recordset
         Dim bizObj As New PubsGetData
         Set rs = bizObj.GetTheData

         TagContents = _
         "<TABLE WIDTH=75% BORDER=1 " _
         & "CELLSPACING=1 CELLPADDING=1>"
         TagContents = TagContents & vbCrLf & vbCrLf
         While Not rs.EOF
             TagContents = TagContents _
             & Chr(9) & "<TR>" & vbCrLf
             TagContents = TagContents _
             & Chr(9) & Chr(9) & "<TD>" & rs("pub_id") _
             & "</TD>" & vbCrLf
             TagContents = TagContents _
             & Chr(9) & Chr(9) & "<TD>" _
             & rs("pub_name") & "</TD>" & vbCrLf
             TagContents = TagContents _
             & Chr(9) & Chr(9) & "<TD>" _
             & rs("city") & "</TD>" & vbCrLf
             TagContents = TagContents _
             & Chr(9) & "</TR>" _
             & vbCrLf & vbCrLf
             rs.MoveNext
         Wend
     End Select

     TagContents = TagContents & "</table>"

 End Sub

 Private Sub Publishers_Respond()
    Publishers.WriteTemplate
 End Sub

 Private Sub WebClass_Start()
    Set NextItem = Publishers
 End Sub
```

תרשים 3.1 בלוקים של קוד והערות



There is quite a bit of advanced VB in this code. If this is new to you, *don't panic!* I provide quite a bit of detail about VB in the Visual Basic excursions in future chapters. For now, follow the logic of what we're doing;the details can come later.

3

Whenever NextItem is set, the Respond method is called. In this case, it is Publishers_Respond. This code appears on line 41, and we invoke the WriteTemplate method on line 42. This tells VB to write the template file to the browser and call ProcessTag.

In ProcessTag, we respond to the tags in the template with code. Whatever we put into TagContents is injected into the HTML stream in place of the tag. In other words, our template is streamed to the client line by line, and each time a tag is found ProcessTag is called, giving us an opportunity to read the tag and respond appropriately.

Let's examine ProcessTag in some detail. On line 10, we declare a recordset object; on line 11, we create a new business object from the PubsGetData object we created in the previous exercise. On line 12, we call its GetTheData method and assign the resulting recordset to the rs variable we created on line 10.



I added some VB code to create indentation and new lines in the HTML output. This will make explicit that the HTML sent from the WebClass is identical to the HTML sent from ASP.

We then start spitting out HTML. Note on line 14 that we assign to TagContents the string `"<TABLE WIDTH=75% BORDER=1 CELLSPACING=1 CELLPADDING=1>"`. We then add in HTML for the rows, interspersing the strings returned from the recordset. When we're done, TagContents is one long string of HTML. We can set a break point on line 17, as shown in Figure 3.24, and use the *Immediate* window to examine the contents of TagContents.

You can see what is in TagContents by opening the Immediate window (Ctrl-G), as shown in Figure 3.25.

Note, however, that this is produced within a loop. The next time through we ***add*** to the TagContents string, building up the table as we go. By the time the loop is finished and we hit the final line (where we add the closing tag for the table), we have built up a complete HTML string for output to the browser.

תרשים 3.1 בלוקים של קוד והערות (המשך)

הסגנונות שבשימוש על ידי Word נשמרים בקובץ ה-HTML כתכונות של התגית <p>, למשל, <p class=c2>. Class זהו כינוי CSS (Cascading Style Sheet) לסגנון, המוגדר בראש דף האינטרנט בתוך תחום ההגדרות של סגנונות, על ידי התגית המכולה <style>. שוב, אנו רואים כאן מידע על תצוגה במקום על מבנה, למרות שהמבנה מרומז (Implicit).

המעבר מ-HTML ל-XHTML נתן בידינו מסמך HTML תואם XML, כפי שראינו בפרק 2. מעבר זה לא פסל אף תגית, הוא רק הבטיח שכולן חוקיות עבור XML, כלומר, שאינן חופפות, ושנעשה שימוש במרכאות כראוי. הסגנונות שנלכדו ב-HTML שרדו במעבר ל-XHTML כמוצג בתדפיס 3.4

תדפיס 3.4 XHTML

```
1: Function GetTheData() As Recordset
    </p>
    <p class="C2">
2:
     <span style="mso-spacerun: yes">

     </span>
Dim rs As ADODB.Recordset
    </p>
    <p class="C2">
3:
     <span style="mso-spacerun: yes">

     </span>
Set rs = New Recordset
    </p>
    <p class="C2">
4:
     <span style="mso-spacerun: yes">

     </span>
Call rs.Open(&quot;select * from publishers&quot;, &quot;dsn=pubs&quot;,
     <span style="mso-spacerun: yes">

     </span>
UID=sa; PWD=;&quot;)
    </p>
    <p class="C2">
5:
     <span style="mso-spacerun: yes">

     </span>
Set GetTheData = rs
    </p>
    <p class="CX">
6: End Function
```

ניתוח

איננו רואים כל שינוי בתגיות <p>; הן עדיין שומרות על הסגנונות, ועדיין אין כל מידע מפורש אודות המבנה, הקושר יחד את הקוד כבלוק של קוד.

למעשה, אנו רוצים לבצע כאן מספר מעברים שונים. ראשית, איננו רוצים לשמור את קודי הסגנונות; במקום זאת אנו רוצים לסמן כל שורה כשורת קוד.

שנית, לא מעניינת אותנו ההבחנה בין השורה האחרונה ובין כל שורת קוד אחרת; ככלות הכל, איננו יודעים כיצד נרצה להציג קוד זה, והשורה האחרונה עשויה להיות מוצגת אחרת מכל שורה אחרת.

שלישית, אנו רוצים לסמן את כל בלוק הקוד, ובכך להפוך את מבנה התדפיס לקשיח. לכשנסיים, קטע זה ייראה כמו הקוד בתדפיס 3.5.

תדפיס 3.5

```
 <code><codeline>
1: Function GetTheData() As Recordset
   </codeline>
          <codeline>
2:
     <spacerun len="4"/>
Dim rs As ADODB.Recordset
   </codeline>
          <codeline>
3:
     <spacerun len="4"/>
Set rs = New Recordset
   </codeline>
          <codeline>
4:
     <spacerun len="4"/>
Call rs.Open("select * from publishers", "dsn=pubs",
     <spacerun len="1"/>
UID=sa; PWD=;")
   </codeline>
          <codeline>
5:
     <spacerun len="4"/>
Set GetTheData = rs
```

```
      </codeline>
            <codeline>
6: End Function
      </codeline>
            </code>
      </code>
```

כל תדפיס הקוד מוקף על ידי תגיות <code>, וכל שורה מסומנת על ידי תגיות <codelines>.

ריווחים וטאבים

ללא ספק הבחנת שבתדפיס המקורי של הקוד, המוצג בתדפיס 3.2, שורות רבות מוזחות. להלן שתי השורות הראשונות:

```
Function GetTheData () As RecordSet
      Dim rs As ADODB . RecordSet
```

הזחה (Identation) מציבה אתגר בפני Word כאשר הוא מתרגם את המסמך ל-HTML. המוסכמה ב-HTML היא לצמצם את כל הרווחים לרווח בודד. Word פותר סוגיה זו על ידי יצירת <span> כמוצג במובאה זו מתוך תדפיס 3.3:

```
<p class=C2>2: <span style="mso-spacerun: yes">      < /span>
```

Word יוצר מרווח, עם התכונה style, שערכה הוא mso (microsoft Office) ו-spaceRun:‏ yes. כך ש-Word שם רווח בודד ואחריו סדרה של תווי רווח ( ), ומסיים את המרווח.

הצהרת המרווח שורדת את המעבר ל-XHTML. בעוד שגישה זו פועלת, XSL לא יציג זאת כהלכה וזוהי תוספת שאינה נדרשת ב-XML. בסך הכל, ב-XML אנו יכולים ליצור תגיות משלנו, ולתת להן תכונות בעלות משמעות. אנו נקדד הזחה באופן כזה שתוכל לשרוד מעברים עתידיים אפשריים.

נתרגם מרווחים אלו לתגית רווח משלנו, עם תכונה המפרטת כמה רווחים נלכדו:

```
<spacerun len="4">
```

תגית זו תציין עבורנו רווח באורך 4 תווים. זה נקי יותר ממרווח ה-HTML של Microsoft, ול-XSL לא תהיה כל בעיה עם זה, שכן זהו XML סטנדרטי.



כפי שצוין קודם, מכיוון שתגית זו היא תגית בודדת, במקום לכתוב:

```
<spacerun len="4"></spacerun>
```

אנו יכולים לדחוס את תגיות ההתחלה והסיום לתגית יחידה על ידי הכללת הלוכסן העוקב:

```
<spacerun len="4"/>
```

סימון Sidebars והערות

פתרנו את הבעיה של צירוף הקוד למבנה בעל משמעות; אנו עומדים בפני אותה בעיה עם "הערה", "**מונח טכני**", "כיצד מבטאים זאת?" ויתר ה-Sidebars.



> לשם פשטות נתייחס לכל ה-Sidebars האלו "הערה", למרות שאתה עשוי לרצות כתרגיל להפריד אותן לתגיות משלהן.

הגישה שלנו ל"הערה", "מונח טכני" וכו' תהיה זהה לזו שנקטנו כלפי הקוד. איננו צריכים לשמר את הסגנון, אבל אנו כן רוצים להפוך את המבנה, שהוא מרומז במסמך ה-Word וה-XHTML, למפורש במסמך ה-XML שלנו.



> הנה הצעדים ל-Visual InterDev. כמוסבר בפרק הקודם, הפרטים המדויקים עשויים להיראות אחרת על המחשב או הרשת שלך, או אם הינך משתמש בטכנולוגיות אינטרנט אחרות.

תדפיס 3.6 מראה כיצד תדפיס זה מופיע בקובץ ה-XHTML:

תדפיס 3.6

```
      <p class="PD">
***Begin Note***
      </p>
      <p class="NO">
Here are the steps for Visual InterDev. As explained in the previous chapter,
➥the exact details may be different on your computer or network,
➥or if you are using different Internet enabling technology.
      </p>
      <p class="PD">
***End Note***
      </p>
```

תדפיס 3.7 מראה כיצד זה ייראה בקובץ ה-XML הסופי:

תדפיס 3.7

```
      <note><noteline>
Here are the steps for Visual InterDev. As explained in the previos chapter,
➥the exect details may be different on your computer or network,
➥or if you are using different internet enabling technology.
           </noteline>
       </note>
```

כל פיסקה בהערה מסומנת בתגית <noteline>, וההערה כולה מוקפת על ידי תגיות <note>.

הפרדת קטעים

למסמך ה-Word, כפי שהוא מוגש על ידי המחבר, אין מבנה מתאר מפורש, למרות שהוא מרומז. למשל, פרק 2 של ספר מחשב מסוים, עשוי להיות מוגש כמוצג בתדפיס 3.8 (גזרתי מספר שורות של טקסט כדי לקצר את התדפיס הזה):

תדפיס 3.8

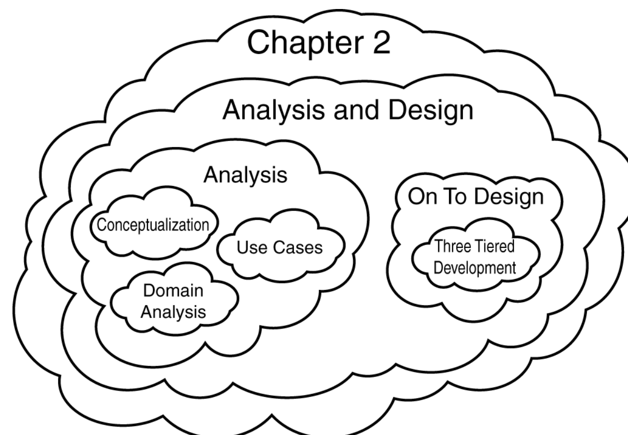
```
0:   (a)2
1:   (b)Analysis and Design
2:   (c)Analysis and Design
3:   (c)Analysis
4:   (d)Conceptualization
5:   (d)use Cases
6:   (d)domain Analysis
7:   (c)On to Design
8:   (d)Three Tiered Development
```

בתוך המסמך הזה ישנו מבנה מרומז. על פי מוסכמה, העורך והמחבר מסכימים שכותרות ברמה-D הם "מתחת" ו"מוכלות על ידי" כותרות ברמה-C. זה הופך למתאר יותר מסורתי, כמוצג בתרשים 3.2.

Chapter 2
Analysis and Design
Analysis
Conceptualization
Use Cases
Domain Analysis
On To Design
Three Tiered Development

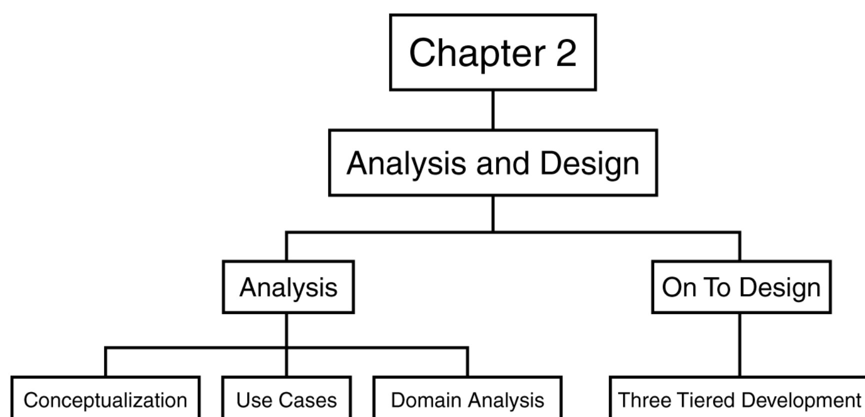
תרשים 3.2 מתאר מסורתי

אנו רוצים שמסמך ה-XML ישקף את מבנה ההכלה (Containment). למעשה, בכל אופן, אנו רואים מבנה עמוק עוד יותר. הקינון במתאר המסורתי משקף למעשה רעיון של הכלה, כמוצג בתרשים 3.3.



תרשים 3.3 הכלה (Containment)

זה קרוב יותר לאופן בו אנו חושבים על המידע. XML הוא די טוב בתפישת הרעיון הזה של הכלה, למרות שהוא עושה זאת במה שנראה כעץ הירכי, כמוצג בתרשים 3.4.



תרשים 3.4 הכלה המוצגת כהירכיה

ישנו מיפוי איזומורפי בין העננים בתרשים 3.3 והעץ בתרשים 3.4.

איזומורפי (Isomorphic) – אחד לאחד. פירושו של יחס **איזומורפי** הוא שלכל אובייקט במקום אחד, ישנו בדיוק אותו אובייקט תואם באחר.

XML מאפשרת לנו ליצור את ההירכיה בתרשים 3.4 על ידי הגדרה מפורשת של רכיבים חדשים כבנים של רכיבים קיימים אחרים. לכן, אנו יכולים ליצור את רכיב הדרגה C, ואז את רכיב הדרגה D, ולהגדיר בפירוש את רכיב הדרגה D להיות רכיב בן של רכיב הדרגה C. למעשה, אנו יכולים לשלוט בסדר הבנים ולהפוך אותו למפורש, כך שרכיבים ברמה-D יהיו ביחסי אחים מסודרים מפורשים אחד עם השני. זה יהפוך לחשוב מאוד כאשר ניצור את טבלת תוכן העניינים הניתנת להרחבה שלנו.

מעבר על ה-HTML הנותר

בסוף התהליך, לאחר שננקה את קובץ ה-XHTML מכל הנ״ל, תיוותר לנו המטלה האחרונה - העברת כל תגיות ה-HTML בעלות המשמעות והטקסט שנותרו לקובץ ה-XML הסופי. אנו צריכים להיות מפורשים אודות אלו תגיות יש להעביר, שכן אנו רוצים להבחין בין תגיות שאנו מזהים ומתכוונים להעביר למסמך הסופי מצד אחד, ובין תגיות שפשוט החמצנו או שאיננו מזהים מצד שני. אנו חייבים ללכוד את התגיות האחרונות שלא נצפו או זוהו, כך שנוכל להרחיב את ה-DTD שלנו ולכלול אותם.

יצירת ה-DTD

עתה, כשאנו יודעים מה אנו מנסים להשיג, אנו יכולים לראות את הדרך להגיע לשם:

1. יצירת DTD המכיל את האילוצים במסמכי ה-XML שלנו.
2. הפיכת מסמך ה-XHTML למסמך XML חוקי.

בהמשך פרק זה נטפל ביצירת ובחינת ה-DTD. ובפרק הבא אנו נראה כיצד לשנות את מסמך ה-XHTML על ידי XSL.

תדפיס 3.9 מראה את ה-DTD במלואו. הבה נבחן אותו שורה אחרי שורה.

תדפיס 3.9

```
<!-- DTD for XWDFS documents -->

<!--
      This is a "macro" that defines normal body text, which
      might contain miscellaneous markup. We define it as an
      entity because we need to reference it from many places
      in the DTD. This mandates that this DTD be referenced
      externally
 -->
<!ENTITY % text "#PCDATA | b | i | u | sub | sup | tab | spacerun | char">

<!-- The root element. Books are made up of one or more sections -->
<!ELEMENT book (section+)>

<!-- Sections have level and id attributes and begin with a title.
Sections may contain other sections -->
<!ELEMENT section (title, (section | div | note | code)*)>
<!ATTLIST section
      level CDATA #REQUIRED
      id CDATA #REQUIRED>

<!-- Titles are just text, which may contain styling markup -->
```

```
<!ELEMENT title (%text;)*>

<!--
      Div elements are our main body content and corresponds
      to paragraphs. Each contains a class attribute, which
      is a copy of the original Word style name
-->
<!ELEMENT div (%text;)*>
<!ATTLIST div
      class CDATA #REQUIRED>

<!--
      Notes and code blocks are our collections of
      non-ordinary text. Each has a encompassing element
      (note, code)and an element for each line/paragraph
      contained in the block (noteline, codeline)
 -->
<!ELEMENT note (noteline+)>
<!ELEMENT code (codeline+)>
<!ELEMENT noteline (%text;)*>
<!ELEMENT codeline (%text;)*>

<!-- The char tag is for special characters -->
<!ELEMENT char EMPTY>
<!ATTLIST char
      type CDATA #REQUIRED>

<!-- special formatting elements to deal with white space -->
<!ELEMENT tab EMPTY>
<!ELEMENT spacerun EMPTY>
<!ATTLIST spacerun
      len CDATA #REQUIRED>

<!-- inline formatting elements,
derived from the familiar HTML tags -->
<!ELEMENT b (%text;)*>
<!ELEMENT i (%text;)*>
<!ELEMENT u (%text;)*>
<!ELEMENT sub (%text;)*>
<!ELEMENT sup (%text;)*>
```

ניתוח

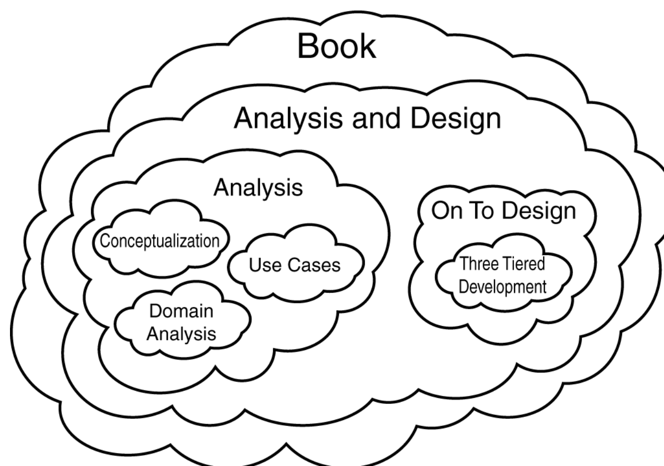
השורות הראשונות הן הערות, ואתה תראה שה-DTD משתמש בתחביר ה-HTML הרגיל של הערות:

<-- הערות כאן --!>.

כל הצהרה ב-DTD מתחילה ב- !> ואחריו מלת מפתח אחת או יותר. ב-DTD שלנו אנו משתמשים בהגדרות ELEMENT או ATTRIBUTE. ב-ELEMENT נשתמש כדי להצהיר על כל הרכיבים ב-DTD , וב-ATTRIBUTE נשתמש כדי להצהיר על התכונות של טיפוס רכיב שנגדיר.

כדי להתחיל הבה נדלג מטה לשורה 12. נחזור לשורה 9 בעוד רגע.

שורה 12 מצהירה על book כעל רכיב. זהו, למעשה, רכיב השורש ל-canoncial form שלנו. מסמכי XML הם תמיד היררכיים. כל רכיב חייב להיות מוכל בתוך ההצהרה של הרכיב ההורה שלו, ואסור שתגיות רכיבים תחפופנה. הרכיב העליון ביותר - במקרה שלנו <book>, משמש כשורש למסמך כולו. אם נחזור לתרשימנו הקודם, בתרשים 3.3, הענן החיצוני ביותר הוא <book>, כמוצג בתרשים 3.5.



תרשים 3.5 הענן החיצוני ביותר השתנה

במסמך ה-XML שמשתמש ב-DTD הזה, אנו נצפה למצוא הוראת DOCTYPE שתתייחס ל-DTD הזה וכן תציין את רכיב השורש: book. שורה 12 ממשיכה ומגדירה את התוכן של רכיב <book>. הוא יהיה מורכב מרכיב section אחד או יותר - כל ספר מורכב מפרק אחד או יותר. סימן הפלוס מציין 1 או יותר. לכן, אנו יודעים שכדי שמסמך BiblioTech יהיה **חוקי** הוא חייב להיות מורכב מרכיב book המכיל לפחות רכיב section אחד.

רכיב Section עצמו מוגדר בשורה 16.

```
16:    <!ELEMENT section (title, (section | div | note | code)*)>
```

שוב אנו רואים הצהרת ELEMENT (רכיב), הפעם ל-Section. כל section, לפי שורה זו ב-DTD שלנו, יורכב מכותרת ומאפס או יותר רכיבים מטיפוס section או div או note או code. הכוכבית (*) מציינת אפס או יותר. הקווים האנכיים בין section, div, note ו-code מראים על היחס or.



הפסיק בהצהרת הקטע מציין סדר. הרכיבים המופרדים בפסיקים חייבים להופיע על פי סדר הופעתם בהגדרה. לכן, הכותרת חייבת להופיע קודם, ואחריה אפס או יותר רכיבים מן הטיפוס המצוין.

מכיון שאין חשיבות (+ או *) עבור title, ה-DTD מסמן 1 בדיוק. שים לב שסימן השאלה (?) יכול גם כן להיות בשימוש בתחביר זה, כדי לסמן 0 או 1.

לכן, לכל רכיב section (פרק/קטע) תהיה תמיד כותרת אחת, ויכולים להיות לו אפס או יותר רכיבים מבין הארבעה שהוזכרו לעיל. שים לב גם כן לכך שאחד מארבעה רכיבים אלה הוא מסוג section גם כן, כך שאנו יודעים שרכיבי section יכולים להכיל רכיבי section אחרים. (פרק בספר יכול להכיל תת-פרקים או קטעים אחרים).

שורה 17 מתחילה את רשימת התכונות לרכיב ה-section. התכונה הראשונה היא level, והיא מטיפוס CDATA. CDATA מייצג טקסט שאינו ניתן לניתוח תחבירי (Parsing): כלומר, מסמך ה-XML יקבל כל מחרוזת טקסט ולא יתייחס אל הטקסט כקוד XML. זה מאפשר לשים תווים שאחרת היו נחשבים אסורים, כמו > או <, המשמשים לייצוג תגיות. לבסוף, התכונה level מסומנת כ-#REQUIRED, המציין שכל section **חייב** את התכונה level כדי להיות חוקי.

#REQUIRED הוא מגביל תכונה (Attribute Modifier). מגבילי תכונה אחרים הם #IMPLIED ו-#FIXED. הראשון, #IMPLIED, מציין שהתכונה היא לא הכרחית ושלא מסופק לה ערך ברירת מחדל. השני, #FIXED, מציין כי רק ערך מסוים יורשה לשימוש בתכונה זו.

התכונה השנייה, id, היא גם כן מטיפוס CDATA וגם היא תכונה נדרשת (Required).

יישויות פרמטרים (Parameter entities)

בשורה 22 אנו מצהירים על אלמנט הכותרת (Title). מסתבר שרכיב הכותרת מכיל אפס או יותר מטיפוסי הרכיבים הבאים: b או i או u או sub או sup או tab או spacerun או char. הוא יכול גם להכיל תווי טקסט, אשר יכולים להיות מנותחים תחבירית. טקסט זה מסומן ב-#PCDATA, המציין Parsed Character DATA - טקסט אשר נדרש לניתוח תחבירי. ההצהרה ל-title צריכה להיות:

```
<!ELEMENT title (#PCDATA | b | i | u | sub | sup | tab | spacerun | char")*>
```

זה יציין שלכל רכיב title יש אפס או יותר רכיבים מתוך תשעת טיפוסי הרכיבים הללו. מסתבר שישנם רכיבים אחרים שהם גם כן מטיפוסים אלו. למעשה, אנו מחשיבים את כל הטיפוסים האלו כטקסט, ולכן יהיה זה שימושי ליצור מאקרו מהיר או קיצור למחרוזת ארוכה זו. כלומר, הרי רכיבים רבים במסמך שלנו ישתמשו בטיפוסים הנ"ל. אז במקום שכל פעם נשתמש בהצהרה הארוכה הנ"ל, ניצור "מילת קיצור" בה נשתמש כשם כולל להגדרת טיפוס ארוכה כזו.

תחביר ה-DTD מאפשר ליצור מאקרוס שכאלו. שני סוגים של מאקרוס כאלו מאפשר לנו תחביר ה-DTD ליצור:

1. ייחוס ליישות (Entity Reference) - החלפת טקסט לשימוש במסמך XML.

ניתן, למשל, ליצור יישות (Entity) להחלפת טקסט. אם אתה רוצה שהטקסט יוחלף בקובץ ה-XML ממש עליך להשתמש בהצהרת יישות פשוטה כמו זו:

```
<!ENTITY LA "Liberty Associates, Inc."
```

שורה זו תאפשר לי להשתמש בייחוס ליישות &LA; במסמך שלי. ובכל מקום בו נשתמש ב-&LA; ה-parser יחליף זאת ב- "Liberty Associates, Inc.". ממש כמו השימוש ב->, ההופך ישירות על ידי הדפדפן ל->, או,   שהופך לתו הרווח.

2. יישות פרמטר (Parameter Entity) – החלפת טקסט לשימוש במסמך DTD.

במקרה שלנו, בכל אופן, אנו לא רוצים שההחלפה תתבצע במסמך ה-XML, אנו רוצים שהיא תתבצע ב-DTD עצמו. לשם כך אנו משתמשים **ביישות פרמטר (parameter entity).**

התחביר ליישות פרמטר הוא להצהיר על היישות עם סימן אחוז (%). לכן ההצהרה על יישות פרמטר ב-DTD תיראה כך:

```
<!ENTITY % LA "Liberty Associates, Inc.">
```

כשנרצה לבצע את ההחלפה, פשוט נקליד %LA; והטקסט "Liberty Associates, Inc." יוכנס לתוך ה-DTD.

בשורה 9 אנו מצהירים על יישות פרמטר שכזו. שם היישות הוא text, ואם אנו כותבים %text; במקום כלשהו ב-DTD, המחרוזת הבאה:

"#PCDATA | b | i | u | sub | sup | tab | spacerun | char" - תושתל במקום.

שורה 22 משתמשת ביישות פרמטר זו, ונראית כך:

```
22:   <!ELEMENT title (%text;)*>
```

והיא שקולה לכתיבה הבאה:

```
22:   <!ELEMENT title (#PCDATA | b | i | u | sub | sup
      ➥| tab | spacerun | char)*>
```

בשורה 29 אנו מצהירים על רכיב חדש, div, המוגדר גם כן להכיל אפס או יותר רכיבים מסוג text (כלומר, אפס או יותר מהרכיבים #PCDATA | b | i | u | sub | sup | tab | spacerun | char). לרכיב div יש גם תכונה נדרשת אחת, שהיא class. Class עצמו מוגדר להיות מטיפוס CDATA.

בשורה 40 אנו מצהירים על רכיב ההערה (Note), והוא מורכב משורת הערה (רכיב Noteline) אחת או יותר, אשר מוגדרות בשורה 41 להכיל אפס או יותר רכיבי text. באופן דומה, בשורה 40, הרכיב code מוגדר להכיל רכיב Codeline אחד או יותר, המוגדרים בשורה 42 להיות מורכבים מאפס או יותר רכיבי text.

הדבר מאוד פורמלי ותמציתי, אך עדיין דרך פירוט המובנת בקלות לגבי מה שאנו יודעים על שורות קוד ושורות הערה. אם נתרגם את שורות 39 ו- 41 לעברית, נאמר ש"הערה מורכבת משורת טקסט אחת או יותר, כאשר טקסט הוא אפס או יותר תווים מנותחים תחבירית (Parsed) ו/או התגיות עבור הדגשה, הטייה, קו תחתון, כתב עילי או תחתי, טאבים, רווחים, או תווים מיוחדים."

ההיבט היחידי שעשוי להיות מבלבל בהגדרה זו הוא הרכיב char (תו). זוהי תגית הנוצרת במיוחד עבור תווים מיוחדים. היא מוגדרת בשורות 45-47.

הרכיב עצמו מוגדר בשורה 45 עם מלת המפתח EMPTY. מלת המפתח EMPTY מציינת שלרכיב אין כל תוכן. במקרה שלנו, זה בגלל שהערך בפועל של התו המיוחד יהיה בתכונה של התגית.

התכונות עבור char מוגדרות בשורות 46 ו- 47. מכאן נלמד כי ל-char תכונה הקרויה type שהיא מטיפוס CDATA והיא תכונה נדרשת (Required).

לכן נצפה למצוא רכיב char שנראה כך:

```
<char type="#018" />
```

בשורה 50 אנו מצהירים על הרכיב tab, ואנו מציינים שהוא תמיד יהיה ריק.

לכן, הכתיבה הבאה היא חוקית:

```
<tab/>
```

או

```
<tab></tab>
```

אך הכתיבה הזו אינה חוקית:

```
<tab>4</tab>
```

זכור, תמצית הרעיון של חוקיות מוגדר על ידי DTD זה. כאשר אנו אומרים "מסמך זה חוקי" אנו מתכוונים שהוא עומד באילוצים של ה-DTD שלו.

בשורות 51-53 אנו מצהירים על הרכיב spacerun (רווח), שהוא גם כן בעל התכונה len, שהיא מטיפוס CDATA והיא תכונה נדרשת.

לבסוף, בשורות 57-61 אנו מצהירים על הרכיבים b, I, u, sub ו- sup, ואנו מצהירים כי כל אחד מכיל אפס או יותר רכיבים מהטיפוס PCDATA# או b או i או u או sub או sup או tab או spacerun או char.

מה ה-DTD אומר לנו

ה-DTD הציג את המבנה של ה-canoncial form שלנו. לכל מסמך שהוא מסוג קנוני תהיה תגית <book> כשורש, אשר תכיל לפחות קטע (Section) אחד.

section יהיה מורכב מכותרת (title) ומאפס או יותר רכיבי section, פסקאות (div), הערות (notes) או בלוקי קוד (code). section, מייצג בדרך כלל פרק בספר.

title יהיה מורכב מאפס או יותר טיפוסי הטקסט השונים (PCDATA, b, i, u, sub, sup, tab, spacerun או char).

b, i, u, sub ו-sup בעצמם יהיו מורכבים מאפס או יותר טיפוסי הטקסט השונים.

div גם כן יהיה מורכב מאפס או יותר טיפוסי הטקסט השונים הללו, אך יש לו תכונה, class, שהיא תכונה נדרשת ומטיפוס CDATA.

הערות (רכיבי note) מורכבות משורת הערה אחת (Noteline) או יותר, כאשר שורת הערה בעצמה היא מטיפוס text. הרכיב code מורכב מאפס או יותר שורות קוד (Codeline), שהם עצמם גם מטיפוסי text.

יש לנו טיפוס תו מיוחד (char), שהוא אחד מטיפוסי הטקסט, והוא תגית ריקה עם תכונת CDATA נדרשת.

טיפוסי הטקסט כוללים גם tab, שהוא רכיב ריק, ו-spacerun, שגם הוא ריק, אך יש לו תכונה נדרשת len.

הצעדים הבאים

כעת, לאחר שאיפיינו את המבנה הקנוני של קבצי ה-XML שלנו, אנו מוכנים להעביר את מסמך ה-XHTML שלנו למסמך XML חוקי, שיעמוד בדרישות ה-DTD שיצרנו זה עתה.

נעשה זאת בשני שלבים: תחילה על ידי שימוש ב-XSL ואחר כך על ידי שינויים ישירים במודל האובייקטים של מסמך ה-XML, ה-DOM של המסמך.

אנו מתחילים עם ה-XSL בפרק 4.

פרק 4
שינוי עם XSL

בפרק זה:

- XSL בפירוט
- הצעדים הבאים

כשבידינו ה-DTD, כל שנותר הוא לעבור ממסמך הXHTML הקיים שלנו למסמך XML חוקי מתאים.

כאמור, אנו נבצע את השינוי בשני שלבים. בשלב הראשון נעבוד עם XSL, ובשני, עם ה-DOM של מסמך ה-XML.

XSL נוצרה כדי לענות על שתי מטרות: הצגת XML וטרנספורמצית XML. המטרה הראשונה, הצגה ועיצוב של XML, מאוד דומה לתפקיד של Cascading Style) CSS (Sheets ב-HTML. המטרה השנייה של XSL – טרנספורמציה – שונה לחלוטין. טרנספורמציה על ידי XSL מאפשרת לנו לקחת קלט קובץ XML וליצור ממנו פלט חדש שהוא קובץ XML שונה במידת מה.

מסתבר ש-IE5 תומך באפשרות הטרנספורמציה של XSL, אך אינו תומך בעיצוב. בשבילנו זה מצוין, שכן ההתמקדות שלנו היא בשינויים במסמך.

בתחילה, אנו נשתמש ביכולות הטרנספורמציה של XSL כדי לשנות את מסמך ה-XHTML שלנו ל-canoncial form שלנו. אחר כך, אנו עשויים לרצות להשתמש שוב ב-XSL כדי לעבור מה-canoncial form שלנו לתצורת תצוגה, אולי אף חזרה ל-HTML. נראה שכיום, למרבית המעצבים לאינטרנט, XSL משמשת בעיקר להפיכת XML ל-HTML לשם תצוגה בדפדפן. בהמשך הספר נראה יישום כזה.

קלטים ופלטים

דפי XSL נכתבים ב-XML, למרות שעשוי להיות להם תחביר פנימי משלהם. העבודה עם XSL מושגת על ידי הזנת מעבד גיליונות הסגנונות (במקרה שלנו MSXML) בשלושת מסמכי ה-XML הבאים:

1. מסמך ה-XML המקורי (במקרה שלנו, מסמך ה-HTML)
2. גיליון סגנונות של XSL הבנוי היטב (זכור כי זהו מסמך XML)
3. מסמך XML לפלט - אופציונלי

הצעד המרכזי הוא בקריאה לשיטה transformNodeToObject על ה-DOM של הקלט. שיטה זו מקבלת שני פרמטרים: DOM עבור גיליון הסגנונות של ה-XSL, ופלט של DOM ה-XML.

אנו נתחיל ביצירת רכיב קטן, הקרוי XSLTransform, שישמש כמעטפת ל-transformNodeToObject. זה יהיה התפקיד של XSLTransform ליצור ולהכין את שלושת קבצי ה-DOMs, כך שנוכל לציין את מסמך הקלט על ידי URL, שם קובץ, או על ידי העברת אובייקט או על ידי העברת מחרוזת ה-XML עצמה.

צעדינו יהיו כדלקמן:

1. יצירת מופע של XSLTransform.
2. ציון שם הקובץ על ידי שימוש במאפיין InputFile של XSLTransform והעברת שם קובץ הקלט. כתוצאה מכך, XSLTransform יקרא קובץ זה וייצור את ה-DOM שלו.
3. ציון שם הקובץ על ידי שימוש במאפיין XSLFile של XSLTransform והעברת שם קובץ ה-XSL. כתוצאה מכך, XSLTransform יקרא את קובץ ה-XSL וייצור את ה-DOM שלו.
4. קריאה לשיטה Transform על XSLTransform, אשר תיצור DOM ריק כפלט, ותקרא ל-transformNodeToObject על DOM הקלט, תוך העברת ה-DOM של XSL שיצרנו זה עתה ו-DOM הפלט החדש כפרמטרים.

קריאה לשיטות (Methods)

כדי להתחיל, אנו חוזרים לקובץ control.asp שהוזכר בפרק 2. הפעם נסתכל על מה שקורה כאשר לוחצים על הכפתור "XHTML to XML".

נזכיר כי מספר משתני מצב (State Variables) נשמרים בתוך עוגיות (Cookies). הם כוללים את מיקום התיקיות של התוכניות והקבצים, כמו גם את ה-DSN והסיסמה לגישה למסד הנתונים.

כאשר המשתמש לוחץ על כפתור, הדף נשלח אל עצמו, (זהו טופס ללא ערך action, לכן נתוני הטופס נשלחים שוב לאותו עמוד. טכניקת ASP זו ידועה בשם Page ReEntry) והתסריט מסתעף בהתאם לכפתור המסוים שנלחץ, כפי שניתן לראות בתדפיס 4.1.

תדפיס 4.1

```
	select case Request("Cmd")
		case "Word to XHTML"
			Response.Write Word2XHTML()

		case "XHTML to XML"
			Response.Write XHTML2XML()
```

כפי שניתן לראות בתדפיס המודגש לעיל, במקרה זה, בו המשתמש לחץ על הכפתור "XHTML to XML" אנו נקרא לפונקציה ()XHTML2XML, המוצגת בתדפיס 4.2.

תדפיס 4.2

```
Function XHTML2XML()
	'convert XHTML file to our canonical XML form
	dim oXSL, oH2X, xmlFN, fn0, dtdFN

	set oXSL = Server.CreateObject("FromScratch.XSLTransform")

	'convert xhtml to xml via a stylesheet
	oXSL.InputFile = dataPath & ".xhtml"
	oXSL.XSLFile = fso.BuildPath(codeDir, "WordXHTML2XML.xsl")

	oXSL.Transform

	'save output as .xml0
	xmlFN = dataPath & ".xml"
	fn0 = xmlFN & "0"
	oXSL.SaveOutputAsFile fn0

	'we assume that the dtd is in the same directory as the data
	dtdFN = "canon.dtd"

	'now pipe the .xml0 to the VB component to do the rest
	➥of the transformation
	set oH2X = Server.CreateObject("FromScratch.WordXHTML2XML")
	oH2X.Convert fn0, xmlFN, dtdFN

	'and we no longer need the intermediate file
	fso.DeleteFile fn0

	XHTML2XML = "Converted " & dataPath & ".xhtml" & " to " & xmlFN
End Function
```

בשורה 124, אנו יוצרים את האובייקט XSLTransform של התיקיה FromScratch, ומשייכים אותו למשתנה oXSL.

הצעד הבא הוא ליצור את ה-DOM עבור קובץ הקלט. בשורה 127 אנו מציבים למאפיין myInputFile באובייקט שלנו את המחרוזת המכילה את שם קובץ הקלט כמוצג בתדפיס 4.3.

תדפיס 4.3

```
0:  Property Let InputFile(ByVal newValue As String)
1:      Dim fso As New Scripting.FileSystemObject, f  As TextStream, xml As String
2:
3:      'read the contents of the file
4:      'should we do this via DOM.load()?
5:
6:      myInputFile = newValue
7:      Set f = fso.OpenTextFile(myInputFile)
8:      xml = f.ReadAll
9:      f.Close
10:
11:     Set myInputDOM = ParseIntoDOM(xml, False, "Input from File")
12: End Property
```

אנו מעבירים את שם הקובץ כמחרוזת.



אם במקום זאת היה לנו URL עבור שם הקובץ, היינו קוראים ל-InputURL. מטרת הרכיב XSLTransform היא רק להפוך את יצירת המופעים של ה-DOMs השונים, הנדרשים למעבר, קלה יותר.

בשורה 7 אנו פותחים את הקובץ עם השם שסופק כפרמטר, ובשורה 8 אנו קוראים את התוכן של קובץ זה לתוך מחרוזת שכינינו בשם xml. אחר כך אנו סוגרים את הקובץ ומעבירים את המחרוזת xml לשיטת העזר שלנו ParseIntoDOM, המוצגת בתדפיס 4.4.

תדפיס 4.4

```
'worker function to read an input source into a new DOM
'if isURL is true, then data is the text of a URL; else data is an XML string
Private Function ParseIntoDOM(data As String, isURL As Boolean, src As String)
➥As DOMDocument
    Dim dom As New DOMDocument, result As Boolean
    dom.async = False
    If isURL Then
        result = dom.Load(data)
    Else
        result = dom.loadXML(data)
    End If
    If Not result Then
```

```
11:         ReportParseError src, dom
12:     End If
13:     Set ParseIntoDOM = dom
14: End Function
```

ParseIntoDOM מקבלת שלושה פרמטרים: המחרוזת המכילה את המסמך, ערך בוליאני של true או false המציין האם שולחים URL או לא, ואת המחרוזת "Input from File", בה נשתמש לדווח על שגיאה, במידת הצורך.

מכיון שלא העברנו URL, ערכו של המשתנה isURL הוא false, הצהרת ה-if בשורה 5 נכשלת והצהרת ה-else בשורה 8 מופעלת. אנו קוראים ל-loadXML, ומעבירים את המחרוזת של כל המסמך, שקראנו קודם.

זה טוען את ה-DOM לזיכרון, ובונה את מודל האובייקטים מהמחרוזת שיצרנו על ידי קריאת המסמך מהדיסק. כשנסיים, יהיה בידינו DOM מתוך קובץ הקלט.

טעינת ה-XSL

הבה נחזור לתדפיס 3.10 בו ניתן לראות שהפעולה הבאה בקובץ ה-ASP שלנו, בשורה 120, היא לקרוא לשיטה XSLFile, הפעם תוך העברת הנתיב של מסמך ה-XSL.

```
120:  oXSL.XSLFile = fso.BuildPath(codeDir, "WordXHTML2XMl.xsl")
```

הפקודה קוראת ל-XSLFile שבתדפיס 4.5.

תדפיס 4.5

```
0: Property Let XSLFile(ByVal newValue As String)
1:     Dim fso As New Scripting.FileSystemObject, f As TextStream, xml As String
2:
3:     myXSLFile = newValue
4:     Set f = fso.OpenTextFile(myXSLFile)
5:     xml = f.ReadAll
6:     f.Close
7:
8:     Set myXSLDOM = ParseIntoDOM(xml, False, "XSL from file")
9: End Property
```

שוב, אנו קוראים ל-ParseIntoDOM, והתוצאה היא יצירת ה-XSL DOM בזיכרון.

בנקודה זו, אנו מוכנים להתחיל בטרנספורמציה. אין לנו עדיין DOM לפלט, אך נטפל בכך בעוד רגע. בשורה 122 בתדפיס 3.10 נמצאת הקריאה הבאה לשיטה Transform:

```
8:      oXSL.Transform
```

הפקודה קוראת לשיטה Transform של האובייקט שלנו, המוצגת בתדפיס 4.6.

תדפיס 4.6

```
0: 'the function that does the work - actually transforms input via XSL to output
1: Public Sub Transform()
2:     Set myOutputDOM = New DOMDocument
3:     myInputDOM.transformNodeToObject myXSLDOM, myOutputDOM
4:     If myOutputDOM.parseError.errorCode <> 0 Then _
4a:        ReportParseError "Output", myOutputDOM
5: End Sub
```

כמובטח, בשורה 2 DOM הפלט נוצר על ידי קריאה ל-DOMDocument חדש. בשורה 3, אנו קוראים לשיטה transformNodeToObject על DOM הקלט. שיטה זו מקבלת שני פרמטרים: XSL DOM ו-outputDOM, וזה בדיוק מה שאנו מספקים לה.

התוצאה המיידית היא שעתה יש לנו שלושה DOMs קיימים: קלט, XSL ופלט. הקריאה ל-transformNodeToObject תבצע את עבודת הקריאה של ה-XSL ושינוי DOM הקלט בהתאם, תוך מיקום התוצאות ב-DOM הפלט.

שים לב כי בשורה 4 אנו בודקים parseError (מייצג שגיאה בניתוח תחבירי), ומדווחים על כך במידה ונמצאת שגיאה כזו. errorCode בעל ערך 0 מציין שלא היו שגיאות.

מטרת המודול XSLTransform היא רק ליצור את ה-DOMs הנחוצים ולקרוא לשיטה המתאימה (transformNodeToObject) תוך העברת הפרמטרים המתאימים. בנקודה זו מנתח תחביר ה-XSL משתלט ויתר העבודה היא אוטומטית.

במידה ומסמך ה-XSL בנוי היטב וחוקי, והחוקים והפקודות שבתוכו נכונים, לא נידרש לבצע צעדים נוספים; מנתח התחביר יפלוט את ה-XML המתאים עם כל השינויים הנדרשים. בכל מקרה אחר, הרכיב ידווח על השגיאה.



MSXML עצמו מבצע עבודה די טובה בדיווח על שגיאות, אך שלא כמו מהדר, הוא מדווח רק על השגיאה הראשונה, ועבורנו כל השגיאות הן קריטיות.

XSL בפירוט

הכנו את קבצי הקלט והפלט שלנו, וקראנו ל-XSL לתוך הזיכרון. עכשיו, בזמן שקוראים ל-transformNodeToObject נותר לנו להתרווח ולתת ל-XSL לבצע את העבודה.

תדפיס 4.7 מציג את כל מסמך ה-XSL. בשארית הפרק, נעבור עליו צעד אחר צעד.

```
<?xml version="1.0"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/TR/WD-xsl"
  xmlns="http://www/w3/org/TR/REC-xml"
  result-ns="">

  <!-- default rules -->
  <xsl:template match="text()"><xsl:value-of/></xsl:template>

  <xsl:template match="/">
        <xsl:pi name="xml">version="1.0"</xsl:pi>
        <xsl:apply-templates />
  </xsl:template>

  <!-- catch any unknown tags -->
  <xsl:template match="*">
        <unknown><xsl:attribute name="tag"><xsl:node-name/></xsl:attribute>
              <xsl:apply-templates />
        </unknown>
  </xsl:template>

        <!-- our top level tag -->
  <xsl:template match="html">
        <html>
              <xsl:apply-templates/>
        </html>
  </xsl:template>

  <!-- tags we can ignore -->
  <xsl:template match="head|title|style|body|script|
  ➥a|meta|link|div"><xsl:apply-templates/></xsl:template>

  <!-- p contain the bulk of the content. we'll map them to div
  ➥and include their class attribute -->
  <xsl:template match="p">
        <div><xsl:attribute name="class"><xsl:value-of
        ➥select="@class"/></xsl:attribute>
        <xsl:apply-templates/>
        </div>
  </xsl:template>

  <!-- convert all code elements to one tagname -->
  <xsl:template match="p[@class='C1'] | p[@class='C2'] | p[@class='CX']">
```

```
        <codeline><xsl:apply-templates/></codeline>
</xsl:template>

<!-- special tag for all paras that are part of notes or sidebars -->
<xsl:template match="p[@class='NO'] | p[@class='SB']">
        <noteline><xsl:apply-templates/></noteline>
</xsl:template>

<!-- just pass special chars along - include all their attributes-->
<xsl:template match="char">
        <char><xsl:attribute name="type"><xsl:value-of
        ➥select="@type"/></xsl:attribute></char>
</xsl:template>

<!-- handle vanilla HTML-like tags - just map them to the same thing,
➥ignoring any attributes which may be present -->
<xsl:template match="b|i|sub|sup|u|br">
        <xsl:element><xsl:apply-templates/></xsl:element>
</xsl:template>

<!-- Word puts in a bunch of <o:p> tags (with no content) inside the
➥"real" <p> tags. These tend to mess up the
        HTML rendering, so we'll take them out. This looks for any p nested inside
        ➥another p and ignores it.  -->
<xsl:template match="p//p">
</xsl:template>

<!-- Word outputs span tags for a bunch of different purposes.
➥Here we handle all the subcases
                that have a style attribute -->
<xsl:template match="span[@style]">
        <xsl:choose>
                <!-- Word's way of outputting a tab - we'll convert to a special tag -->
                <xsl:when expr="getStyleName(this) == 'mso-tab-count'">
                        <tab />
                </xsl:when>

                <!-- How Word encodes consecutive spaces - since HTML would
                ➥swallow them up
                        we change to a tag that denotes the number of spaces -->
                <xsl:when expr="getStyleName(this) == 'mso-spacerun'">
                        <spacerun><xsl:attribute name="len"><xsl:eval>
                        ➥CountOccurrences(this.firstChild.nodeValue,
                        ➥'\xa0')</xsl:eval></xsl:attribute></spacerun>
```

```
			</xsl:when>

			<!-- we ignore these -->
			<xsl:when expr="getStyleName(this) == 'mso-bookmark'">
				<xsl:apply-templates />
			</xsl:when>

			<!-- we'll also ignore local font changes -->
			<xsl:when expr="getStyleName(this) == 'font-family'">
				<xsl:apply-templates />
			</xsl:when>

			<xsl:when expr="getStyleName(this) == 'font-style'">
				<xsl:apply-templates />
			</xsl:when>

			<!-- can't even figure out what this is, so we'll ignore it -->
			<xsl:when expr="getStyleName(this) == 'layout-grid-mode'">
				<xsl:apply-templates />
			</xsl:when>

			<!-- flag any other cases, recording their tagname
			➥and style so we can debug our stylesheet -->
			<xsl:otherwise>
				<unknown>
					<xsl:attribute name="tag">span</xsl:attribute>
					<xsl:attribute name="val"><xsl:eval>
					➥getStyleName(this)</xsl:eval></xsl:attribute>
					<xsl:apply-templates />
				</unknown>
			</xsl:otherwise>
		</xsl:choose>
</xsl:template>

<xsl:script>
<![CDATA[
// return number of c characters in the string s
function CountOccurrences(s, c)
{
	var n = 0;
	for(var i = 0; i < s.length; i++)
	{
		if (s.charAt(i) == c) n++;
	}
```

```
118:         return n;
119:  }
120:
121:  // return the part of the style attribute's value before the colon
122:  function getStyleName(me)
123:  {
124:         var styleArg = me.getAttribute("style");
125:         var p = styleArg.indexOf(':');
126:         return styleArg.substr(0, p);
127:  }
128:  ]]>
129:  </xsl:script>
130:  </xsl:stylesheet>
```

ניתוח

Namespace

השורה הראשונה בתדפיס 4.7 מצהירה כי גיליון סגנונות זה נכתב בהתאם לגירסה 1.0 של XML. זכור כי דף XSL נכתב ב-XML הבנוי היטב.

שורה 2 מצהירה על גיליון הסגנונות, וכחלק מהצהרה זו מקימה שלושה **namespaces**. הרעיון של namespaces אינו ייחודי ל-XML; ניתן למצוא namespaces ב- C++ ובמיגוון שפות אחרות.

הרעיון של namespace הוא לאפשר ליצור תגיות בעלות משמעות מבלי לדאוג שמא שם של תגית שתבחר עשוי להתנגש בשם תגית של ארגון אחר. W3C הכריז על מספר namespaces שמורים, ובכללם namespaces עבור HTML, XSL, ו-XML.

מסתבר שזה מאוד עוזר בזמן קריאת גיליון הסגנונות. ה-namespace עוזר לנו להבחין בין התגיות הייחודיות ל-XSL ואלו שהן XML.

בשורה 2 אנו מצהירים שאנו נשתמש ב-Namespace בשם XSL (התחביר הוא xmlns:xsl), אותו אנו קובעים למפרט המצוי ב-URL הבא: http://www.w3c.org/TR/WD-xsl. שים לב כי ה-URL עשוי שלא להצביע על מסמך בעל משמעות. מה שהכרחי הוא ש-URL זה יהיה ייחודי. למעשה, ה-URL משמש כמזהה ייחודי גלובלי.

אנו רוצים להצהיר על ה-namespace של XSL כזה המצוי ב-URL http://www.w3c.org/TR/REC-xml. עם זאת, שים לב כי התחביר בשורה 3 שונה. כאן אנו מצהירים על namespace **ללא שם**. זה מראה שאם אנו לא מציינים בפירוש namespace בתגית, ניתן להניח שמדובר ב-namespace של XML.

לבסוף, בשורה 4, אנו קובעים את ערך המאפיין result-ns. זהו מאפיין סטנדרטי של הרכיב xsl-stylesheet והוא משמש לציון הפלט של מעבד ה-XSL. במקרה שלנו, הגדרנו אותו כמחרוזת ריקה, ובכך ציינו את ה-namespace ללא שם: במקרה זה XML. מסתבר שב-IE5 כל פלט הוא XML, ומתעלמים ממאפיין זה!

תגית הפתיחה של רכיב גיליון הסגנונות משתרעת על פני 4 שורות ומסתיימת בשורה 4. תגית הסגירה לגיליון הסגנונות היא בשורה 130.

תבניות (Patterns-ו templates) ופילטרים

מטרתנו בשארית גיליון הסגנונות של XSL היא לחפש אחר רכיבים במסמך המקור שלנו (מסמך ה-XHTML) לטפל בהם ולפלוט אותם למסמך היעד (מסמך ה-XML החדש).

למרבית המתכנתים הקושי ב-XSL טמון בכך שהיא שפה הצהרתית ולא פרוצדורלית. כלומר, במסמך XSL מצהירים על כל הרכיבים שרוצים למצוא (באמצעות שימוש בהתאמת תבניות) והפעולה שיש לנקוט על רכיבים אלו. צריך מעט זמן להתרגל לכך. הרבה יותר קל להבין זאת בדוגמה.

כדי להתחיל, הבה נדלג מטה לשורה 9, כדי לבחון דוגמה פשוטה יחסית של התאמת רכיב ונקיטת פעולה. הדבר שאנו מעוניינים להשיג כאן הוא הפניית מעבד ה-XSL לעיבוד כל הרכיבים מצומת השורש ומטה. נבצע זאת באופן הבא:

```
 9:  <xsl:template match="/">
10:      <xsl:pi name="xml">version="1.0"</xsl:pi>
11:      <xsl:apply-templates />
12:  </xsl:template>
```

בשורה 9, אנו רואים את תגית ה-template ב-namespcae הקרוי xsl. כתיבת :xsl מציינת את הXSL Namespace-, ולכן, בהתבסס על הצהרת ה-namespcae בראשית הקובץ, אנו יודעים שרכיב template הוא רכיב XSL.

תבנית (Template) היא רכיב XSL המציין תבנית לפיה יש לבצע התאמה ב-DOM הקלט.

תבנית המטרה (Pattern) מסומנת במקרה זה בתכונה של ההתאמה (match). במקרה זה, התבנית היא "/", מסתבר כי היא תבנית מיוחדת המתאימה לצומת השורש.

צומת השורש הוא רכיב מרומז בכל מסמך XML: זו נקודת ההתחלה הגבוהה ביותר במסמך.

ערך התבנית (Template) מופיע, כמו בכל התגיות, בין התגית הפותחת לסוגרת. במקרה זה ערך התבנית הוא בשורות 10 ו- 11. כאשר רכיב במסמך הקלט מתאים לתבנית, ערך התבנית יועבר לקלט.

בשורה 10 אנו יוצרים פקודת עיבוד (Processing Instruction) של XML, ששמה הוא xml, וערכה הוא "version="1.0. ניתן לראות פקודת עיבוד זו בראש כל קובץ XML, ואנו נוסיף אותה לשלנו גם כן.

בשורה 11 מוצאים את רכיב ה-apply-templates XSL, אשר גורם למנתח תחביר ה-XSL לפעול ברקורסיה על הצומת הנוכחי. הצומת הנוכחי הוא הצומת שהתאים לתבנית ברכיב xsl:template בשורה 9.

במקרה שאנו בוחנים, הצומת הנוכחי הוא צומת השורש. בקריאה הנוכחית ל-apply-templates לא צוינו בנים כלשהם (שכן אחרת היו מצוינים על ידי התכונה select). כשלא מצוינים בנים, המעבד ינסה למצוא תבניות מתאימות ל**כל** הבנים של הצומת הנוכחי.

בשורה 12 הצומת xsl:template נסגר, שכן כל רכיבי ה-XML חייבים להיסגר. התוצאה של שורות 9-12 גורמת למנתח התחביר לחפש אחר התאמות לכל צומת במסמך הקלט.

הבה נחזור אחורה לשורה 7:

```
7:      <xsl:template match="text()"><xsl:value-of/></xsl:template>
```

כאן אנו רואים תבנית (Template) נוספת. הפעם התכונה match נקבעת ל-"text()". זה יתאים לכל צומת טקסט. הערך של תגית ה-template מוכתב על ידי הרכיב xsl:value-of, המחזיר את ערך הצומת התואם כמחרוזת. התוצאה היא העתקת הצומת התואם למסמך הפלט. שים לב כי תגית ה-template נסגרת, שכן כל תגיות ה-XML צריכות להיסגר.

כך כל צמתי הטקסט במסמך המקור (ה-XHTML) יועתקו כפי שהן למסמך הפלט. זוהי צורת ההתנהגות הטבעית של XSL, אך אנו יצרנו עבורה כלל, שכן IE5 לא עושה זאת באופן אוטומטי.

עד עתה עברנו על 13 השורות הראשונות של דף ה-XSL. כל השאר לפי גישה בסיסית זו של התאמת רכיבים במסמך המקור ונקיטת הפעולה המתאימה.

התאמת תבניות (Templates)

דרך אחת לחשוב על מסמך XSL היא לדמיין ממיין מטבעות, כפי הנראה בתרשים 4.1.

המטבעות מוכנסים פנימה לממיין המטבעות ומתגלגלים מטה במורד השיפוע עד אשר הם מוצאים פתח שדרכו הם יכולים לעבור. כשהם עוברים על פני חריץ מספיק גדול, הם נופלים לתוך הצינור הימני. בעת יצירת ממיין מטבעות שכזה חייבים לשים לב לסדר החריצים. אם תשים חריץ גדול לפני חריץ קטן יותר, הרי שמטבעות קטנים יותר יפלו לתוך הצינור הגדול יותר, ואנו לא רוצים שזה יקרה. לכן, החריצים חייבים להיות ממוינים מהקטן ביותר לגדול ביותר. הסדר אינו מוכתב על פי ערכם היחסי אלא על פי גודלם היחסי.

באופן דומה, התוכן של מסמך המקור שלנו מתגלגל במורד שיפוע ממיין, כאשר אנו חייבים לבדוק את המקרים המפורטים ביותר לפני שנבדוק את הכלליים יותר. סך הכל, אם יש לנו חריץ המקבל "כל רכיב" הרי שכל הרכיבים יפלו דרך חריץ זה ולעולם לא יגיעו לתבנית מפורטת יותר המתאימה להם.

השיפוע במקרה זה הוא מתחתית המסמך כלפי מעלה. כלומר, IE5 בודק את התבנית (Pattern) התחתונה ביותר במסמך לפני שהוא בודק את זו שמעל, והוא ממשיך ב**מעלה** מסמך ה-XSL עד שהוא מוצא התאמה טובה.



תרשים 4.1: ממיין מטבעות

סיור קצר

המיון מלמטה למעלה הוא מוסכמה הייחודית למיקרוסופט. המלצת W3C אומרת שכשישנן מספר התאמות, "... כל כללי התבניות המתאימות שהם פחות חשובים מהכללים ומכללי התבניות המתאימות החשובים ביותר לא נלקחים בחשבון."

כלומר, המפרט אומר כי אם ישנה יותר מתבנית מתאימה אחת, ההתאמה היא רק לחשובה ביותר.

החשיבות נמדדת על פי מיקום התבנית. ניתן לייבא גיליונות סגנונות לתוך גיליון הסגנונות ה"ראשי", אך תבניות שבגיליונות סגנונות מיובאים חשובים פחות מאלו שבגיליון הסגנונות הראשי.

המפתח יכול גם לקבוע סדר חשיבות עבור הכלל של תבנית נתונה. העדיפות יכולה להתבטא על ידי מספר שלילי או חיובי. ערך ברירת המחדל הוא 0. על ידי קביעת העדיפויות, המפתח חופשי להחליט אלו כללים הם החשובים ביותר.

IE5 אינו תומך במושג החשיבות, וגם לא בייבוא של גיליונות סגנונות.

מכיון ש-IE5 יעריך את ההתאמות מתחתית הקובץ עד לתחילתו, נשים את ההתאמות הכלליות ביותר בקרבת תחילת הקובץ. זה מבטיח שנתפוס את כל הרכיבים שאחרת לא היו נמצאים מתאימים.

מציאת תגיות לא ידועות

אנו רוצים שגיליון הסגנונות של XSL שלנו יקח בחשבון כל תגית במקור שלנו (מסמך ה-XHTML). כל תגית צריכה להיות מזוהה, ומעובדת בצורה כלשהי, ולא רק כדי להיות מועברת בשלמות למסמך הפלט.

כחלק מתהליך איתור השגיאות, אנו צריכים למצוא את כל התגיות שלא היינו מצפים למצוא בקובץ. נסמן תגיות מלאכותיות אלו כ״לא ידועות״, כך שנוכל לאתר אותן מהר בקובץ הפלט ולהחליט מה לעשות איתן.

למשל, נניח שישנה תגית "xyz" ב-XHTML. כמו כן, נניח כי אין לנו כל מושג לגבי משמעות התגית, ולכן לא טיפלנו בה בגיליון הסגנונות. אנו רוצים שהפלט יכלול:

```
<unknown tag = "xyz"/>
```

אם לתגית <xyz> יש תוכן, נרצה לראות זאת בפלט גם כן. כך, שאם במקור קיים:

```
<xyz>Cogito Ergo Sum</xyz>
```

אנו רוצים שהפלט יכלול:

```
<unknown tag="xyz">Cogito Ergo Sum</unknown>
```

הנה האופן בו נעשה זאת. בתחילה, אנו יוצרים בתחילת הקובץ ״כלל התאמה לכל רכיב״:

```
15:    <xsl:template match="*">
```

ה-* מציינת ״כל רכיב״. זכור, אם זה בתחילת הקובץ תהיה התאמה רק אם כל שאר ההתאמות ייכשלו.



* מתאימה רק לרכיבים, ולכן אינה מתאימה גם לצומת השורש, צמתי טקסט, הערות, הוראות עיבוד, וכל שאר הצמתים שאינם רכיבים.

כשתושג התאמה שכזו, ניצור תגית <unknown>, ונשייך תגית attribute לאותה תגית <unknown>. נשיג זאת על ידי הרכיב xsl:attribute, המקבל כתכונה **שלו** את שם התכונה שמשייכים לרכיב הנוכחי.

משפט זה מעט קשה לעיכול, לכן נסביר אותו באמצעות דוגמה. אם אנו יוצרים את התגית <unknown> אנו יכולים לתת לה את התגית attribute על ידי כתיבת:

```
<xsl:attribute name="tag">
```

משמעות הדבר ״לשייך לתגית הנוכחית (<unknown>) תכונה ששמה הוא tag.״

גוף התגית attribute הופך לערך של תכונה זו ב-<unknown>. לכן היינו יכולים לכתוב:

```
<unknown><xsl:attribute name="tag">"xyz"</xsl:attribute>
```

זה אומר ״לשייך לתגית הנוכחית תכונה ששמה הוא tag וערכה הוא xyz.״ התוצאה תהיה:

```
<unknown tag="xyz"/>
```

זה היה עובד פשוט מצוין אם היינו יודעים שהתגית שאנו עובדים איתה היא xyz. לרוע המזל, איננו יכולים לדעת באילו תגיות אנו עשויים להיתקל, לכן אנו חייבים לשאול את מסמך המקור מהו שם התגית הלא ידועה. זאת נעשה באמצעות xsl:node-name:

```
16:    <unknown><xsl:attribute name="tag"><xsl:node-name/></xsl:attribute>
```

יש לקרוא זאת באופן הבא: "צור תגית <unknown> ותן לה תכונה ששמה הוא tag וערכה הוא מה שנמצא ב-xsl:node-name." והרי התוצאה במקרה זה:

```
<unknown tag="xyz"/>
```

לאחר שיצרנו את התגית <unknown>, אנו חייבים לאסוף את כל בניה. אנו עושים זאת על ידי קריאה רקורסיבית ל-xsl:apply-templates, שתגרום למנתח התחביר לבחון את תוכן התגית <xyz>. במקרה שהוצג לעיל:

```
<xyz>Cotigo Ergo Sum</xyz>
```

רקורסיה זו תתאים לתבנית שבחנו בשורה 7:

```
7:    <xsl:template match="text()"></xsl:template>
```

הטקסט ייקבע כצומת-בן של התגית <unknown>, בדיוק כפי שציפינו, והתוצאה הינה:

```
<unknown tag="xyz">Cotigo Ergo Sum</unknown>
```



שים לב שאין לנו שום כלל למעבר באיטרציה על התכונות של צומת זה, משום כך לא נאסוף תכונות כלשהן בתגית הלא ידועה שלנו. אנו פשוט נזרוק אותן. לפיכך, אם הצומת המקורי שלנו היה:

```
<xyz size="3">Cotigo Ergo Sum</xyz>
```

הפלט שלנו לא יכלול את התכונה size, ועדיין יישאר:

```
<unknown tag="xyz">Cotigo Ergo Sum</unknown>
```

שורות 14-19 מציגות את כל הרצף הזה:

```
14:  <! - - catch any unknown tags - - >
15:     <xsl:template match="*">
16:        <unknown><xsl:attribute name="tag"><xsl:node-name/></xsl:attribute>
17:            <xsl:apply-templates />
18:        </unknown>
19:     </xsl:template>
```

הבה נניח שוב כי נתקלנו בתגית <xyz> במסמך המקור שלנו. אנו לא ננסה להתאים את תגית ה-<xyz> הזו לאף שורה מתחת לשורה 15 שכן לא היינו מצפים לתגית זו. אנו כן נתאים אותה לשורה 15, שכן * מתאימה לכל דבר.

אנו יוצרים תגית <unknown> ונותנים לה את התכונה "tag=xyz" כמוצג בשורה 16. אחר כך אנו יכולים לקרוא ל-apply-templates על תגית xyz זו כדי לוודא שאם יש לה בנים, גם הם יעובדו. לבסוף, בשורה 18 אנו סוגרים את התגית הלא ידועה ובשורה 19 אנו סוגרים את התבנית.

בניית ה-DOM

הנה שאלה מעניינת שיש לקחת בחשבון לפני שנמשיך. אם בקובץ המקור היינו מוצאים:

```
<xyz value="foo">
```

מה היה צריך להכיל קובץ הפלט? התשובה היא:

```
<unknown tag="xyz" />
```

כפי שצוין קודם לכן, אנו משליכים את התגית foo, אך כיצד זה קיבלנו תגית הסוגרת את עצמה? הרי לפי המקור שלנו יש תגית סגירה </unknown>. האם אנו לא אמורים לקבל את הפלט הבא?

```
<unknown tag="xyz"></unknown>
```

כדי להבין זאת, יש לזכור מה באמת מתרחש. אנו קראנו את מסמך המקור לתוך DOM. כמו כן, קראנו גם את מסמך ה-XSL לתוך DOM, והפלט שלנו הוא בעצמו DOM. כשאנו יוצרים את התגית <unknown>, אנו לא עושים זאת בתוך קובץ, אלא ב-DOM עצמו. DOM הפלט מכיל עתה תגית unknown שיש לה תכונה tag, בעלת הערך xyz. אם אין אובייקטים בנים (כאשר אין טקסט) אז כשנכתוב את DOM הפלט לקובץ טקסט (למשל Chap2.xml) מנתח התחביר ייצור תגית בודדת, הסוגרת את עצמה:

```
<unknown tag="xyz" />
```

ובמקרה שבו מסמך המקור הכיל:

```
<xyz>Cotigo Ergo Sum</xyz>
```

ל-DOM הפלט יש תגית <unknown> עם התכונה tag שערכה הוא xyz, אך יש לה גם תגית בן מסוג text, שערכה הוא Cotigo Ergo Sum. לכן, כשזה נכתב למסמך הפלט, זה נכתב כך:

```
<unknown tag="xyz">Cotigo Ergo Sum</unknown>
```

כל זה הוא תוצאה אוטומטית מפליטת תוכן DOM הפלט.

ניתוח תחבירי של ה-XSL (Parsing the XSL)

עד כה עברנו על שורה 18 של מסמך ה-XSL שלנו. הצהרת התבנית הבאה היא ישירה למדי:

```
22:  <xsl:template match="html">
23:       <html>
24:            <xsl:apply-template/>
25:       <html>
26:  <xsl:template>
```

אנו מתאימים כל רכיב שבו התגית היא html. במקרים הרגילים נצפה למצוא רק רכיב אחד כזה. היינו יכולים לפסול את התגית HTML בשלב זה; בסך הכל מסמך הפלט XML שלנו לא יהיה מסמך HTML ולכן לא יצטרך את התגית HTML.

מכיון שאנו מבצעים את המעבר ל-canoncial form שלנו בשני שלבים, נבנה קובץ XML זמני. קובץ זה זקוק לצומת שורש, ולעת עתה, נוח יותר להשאיר אותו כ-HTML, כך שאנו פשוט מעתיקים את התגית HTML במקום. במהלך השלב השני של המעבר נחליף את התגית HTML ברכיב השורש החדש שלנו: <book>.

תגיות שניתן להתעלם מהן

שורה 29 מוצאת את כל התגיות במסמך המקור שלא איכפת לנו מהן, ומשליכה אותן:

```
29:     <xsl:template match="head|title|style|body|script|
        ➥a|meta|link|div"><xsl:apply-template/></xsl:template>
```

מבחינת התחביר תבנית זו מתאימה ל-head או ל-title או ל-style וכן הלאה, לאורך כל הרשימה. צריך לקרוא ל-apply-templates, המבטיחה שאם לאחד מן הרכיבים הללו יש בנים, אנו נאסוף אותם. אנו לא נוקטים כל פעולה נוספת, כולל העתקת התגית למסמך הפלט שלנו, כך שבפועל התגית בעצם מושלכת.

טיפול בתגית <p>

בחינת מסמך המקור מראה כי מרבית התוכן מוחזק בתוך תגיות <p>. אין זה מפתיע; מאחר ומרביתו של הספר הוא בתוכן הספר, והטקסט מאוחסן על ידי Word בפסקאות המסומנות על ידי תגיות <p> כאשר Word מייצא ל-HTML.

תגית ה- <p> התפתחה מאז ה-HTML המוקדם, וכעת היא נחשבת לתערובת של רכיב מבנה ורכיב תצוגה. כתוצאה מכך ההתנהגות של תגיות <p> פחות אמינה. החלטנו לשנות את הפסקאות האלו ל- <div>:

```
32:  <xsl:template match="p">
33:      <div><xsl:attribute name="class"><xsl:value-of
         ➥select="@class"/></xsl:attribute>
34:      <xsl:apply-templates/>
35:      </div>
36:  </xsl:template>
```

בשורה 32 אנו מבצעים התאמה לכל רכיבי ה-P. הבה נפרק את שורה 33 לארבעת החלקים שלה:

```
33a:          <div>
33b:  <xsl:attribute name="class">
33c:  <xsl:value-of select="@class"/>
33d:      <xsl:attribute>
```

בחלק (a), אנו כותבים הצהרת <div> לפלט. לכן, בכל פעם שנמצא תגית <p> נחליף אותה ב- <div>.

בחלק (b), אנו משייכים תכונה שנקראת "class" ל-div. לכן ה-div שלנו יראה:

<div class=

בחלק (c), אנו אוספים את הערך של התגית <p>, ובוחרים רק את התכונה class. הסימן @ מסמל תכונה. התוצאה בפועל היא שאנו לוקחים את התכונה class מ- <p>. אם התחלנו עם <p class="FT"> נסיים עם <div class="FT">.

לבסוף, בחלק (d) אנו סוגרים את התבנית שהתחלנו בשורה 32.

צירוף שורות הקוד וההערות

נזכור כי בדיון הקודם רצינו לצרף יחד את כל שורות הקוד. נעשה זאת בשני שלבים. השלב הראשון הוא לאסוף את כל השורות עם הסגנון "C1", "C2" או "C3" ולסמן אותן עם התגית <codeline>. בפרק הבא נראה כיצד הם מצורפים יחד לבלוקי <code>. ההמרה ל- <codeline> מוצגת בשורות 38-41 :

```
38:  <!-- convert all code elements to one tagname -->
39:  <xsl:template match="p[@class='C1'] | p[@class='C2'] | p[@class='CX']">
40:          <codeline><xsl:apply-templates/></codeline>
41:  </xsl:template>
```

כאן ההתאמה שלנו היא לפי התבנית (Pattern). אנו מחפשים אחר רכיבי p שיש להם את התכונה 'class='c1 או שיש להם את התכונה 'class='c2 או 'class='c3. אם נמצא התאמה לאחד מאלו, נכתוב את התגית <codeline>, ואחר כך נקרא ל-apply-templates כדי לקבל את בניהם (במקרה זה, כדי לקבל את הטקסט). אחר כך אנו סוגרים עם תגית הסגירה </codeline>, ובשורה 41 אנו סוגרים עם תגית הסגירה </xsl:template>.

כשאנו יוצרים את DOM הקלט, אנו עשויים להיתקל בשורות הבאות ב-XHTML מפרק 3 :

```
1166:     <p class="C2">
1167: 1: Function GetTheData() AS Recordest
1168:        </p>
```

כאן נוצר רכיב p ב-DOM. רכיב ה-p יכיל תכונה ('class='c2) ויהיה לו בן יחיד : טקסט שערכו הוא : "1: Function GetTheData() As Recordset"

גיליון הסגנונות שלנו יהפוך את רכיב ה-p לרכיב שורת קוד (codeline) בקובץ הפלט. אחר כך הקריאה ל-apply-templates תיכנס ברקורסיה לתוך רכיב ה-p, ותמצא את צומת ה-()text. זה יותאם לתבנית הקודמת :

```
7:      <xsl:template match="text()"><xsl:value-of/></xsl:template>
```

אשר תגרום ל-DOM הפלט ליצור רכיב טקסט עם אותו הטקסט :

```
1:      Function GetTheData() As Recordset
```

ולשייך את הטקסט לרכיב ה-codeline החדש. כשזה נכתב למסמך הפלט, התוצאה תיראה כך :

```
<codeline>
      1:     Function GetTheData() As Recordset
</codeline>
```

אותה תבנית עיצוב בדיוק מיושמת על הערות בשורות 44-46:

```
44:  <xsl:template match="p[@class='NO'] | p[@class='SB']">
45:          <noteline><xsl:apply-templates/></noteline>
46:  </xsl:template>
```

ההבדל היחידי בין הקוד הזה לקודם הוא שכאן אנו אוספים קוד המסומן במחלקה "NO" או "SB". SB מציין סרגלי צד (Sidebars) ולעתים העורכים מעדיפים סרגלי צד מסוימים עם סגנון זה במקום עם NO.

תווים מיוחדים

שורות 49-51 מטפלות בנוכחות של תווים מיוחדים ב-XHTML. בפרק הקודם כבר היה לנו עיבוד מיוחד עבור תווים מסוימים. ישנם שני סוגי תווים המעניינים אותנו. הראשון הוא ציטוטים חכמים (Smart Quotes), אשר תייגנו כ-char בקוד הזה:

```
s = Replace(s, Chr(146), "<char type =" "smartApos" "/>")
s = Replace(s, Chr(147), "<char type =" "smartLQuote" "/>")
s = Replace(s, Chr(148), "<char type =" "smartRQuote" "/>")
```

הסוג השני הוא תווים של הוצאת Macmillan, שאותם תייגנו עם הקוד הבא:

```
s = Replace(s, "[em]", "<char type =" "emSpace" "/>")
s = Replace(s, "[md]", "<char type =" "emDash" "/>")
s = Replace(s, "[lb]", "<char type =" "bullet" "/>")
```

כעת לקובץ ה-XHTML יש מספר רכיבי תווים (char) עם תכונה של type, שערכה אומר לנו איזה סוג של תווים הם. לעת עתה, אנו רק נעביר אותם הלאה. כדי לעשות זאת, אנו חייבים ליצור טיפוס <char> במסמך הפלט ואז לשייך לו את התכונה שנמצאת במסמך המקור:

```
49:  <xsl:template match="char">
50:          <char><xsl:attribute name="type"><xsl:value-of
             ➥select="@type"/></xsl:attribute></char>
51:  </xsl:template>
```

שורה 49 מבצעת התאמות לרכיבי תווים אלו. הבה נפרק את שורה 50 לארבעה חלקים:

```
50a:         <char>
50b:  <xsl:attribute name="type">
50c:  <xsl:value-of select="@type"/>
50d:  <xsl:attribute></char>
```

בחלק (a) אנו כותבים את רכיב ה-<char> ל-DOM הפלט. בחלק (b) אנו משייכים לרכיב זה תכונה ששמה הוא type. בחלק (c) אנו נותנים ערך לתכונה חדשה זו, בפרט את הערך המושג על ידי קריאה ל-value-of ובחירת התכונה type ברכיב המקור. לבסוף, בחלק (d) אנו סוגרים את התכונה שיצרנו ב- 50b ואת רכיב ה-char החדש שיצרנו ב- 50a. בשורה 51 אנו סוגרים את התבנית שהתחלנו בשורה 49.

שלוש השורות הבאות עוברות על רכיבי html מסוימים (b, i, sub, sup, u ו-br) אך מתעלמות מכל תכונה שעשויה להיות להם:

```
<xsl:template match="b|i|sub|sup|u|br">
        <xsl:element><xsl:apply-templates/></xsl:element>
</xsl:template>
```

הלוגיקה היא כמעט זהה, אך התחביר מעט שונה. מכיון שאנו לא יכולים לדעת איזה רכיב אנו יוצרים (הוא עשוי להיות כל אחד מהשישה שמתאימים), אנו משתמשים ב-xsl:element. הוא בונה רכיב פלט אשר על פי ברירת מחדל קרוי על פי שם התגית של הרכיב הנוכחי. לכן, אם מצאנו התאמה עם תגית <b>, אז xsl:element יבנה תגית <b> במסמך הפלט.

אחר כך נקרא ל-apply-templates כדי לאסוף את כל הבנים של הרכיבים האלו. אם במקור יש:

```
<b>This text is bold</b>
```

אז הבן של רכיב b זה הוא רכיב טקסט עם הטקסט: "This text is bold".

ניקוי סימוני פסקאות מקוננות

כפי שצוין בהערות, כאשר word יוצר פלט כדף אינטרנט הוא מפזר סדרה של תגיות <o:p>, חסרות כל תוכן. נדיר למצוא תגיות <p> מקוננות וישנה עדות אמפירית לכך שקיומן יוצר בעיות בהצגת מסמך הפלט (כלומר, ניסינו זאת וזה נראה אווילי). לכן אנו נפטרים מתגיות ה-<p> המקוננות באמצעות הקוד שבשורות 60 ו- 61:

```
60: <xsl:template match="p//p">
61: </xsl:template>
```

הלוכסן הכפול הוא סימן XSL מיוחד המציין "כל הצאצאים". לכן יש לקרוא התאמה זו כ"מצא את כל הצאצאים של p שהן גם כן p" – כלומר, מצא את כל רכיבי ה-p המקוננים.

הפעולה הננקטת? כלום. אנו פשוט סוגרים את התבנית ללא כל תוכן פנימי. כך שלכל תגית כזו יותאם - ערך ריק. התוצאה בפועל, תהיה הסרתן מן המסמך.

אתה עשוי לתהות מדוע למצוא להן התאמה מלכתחילה; האם הן לא ייעלמו אם פשוט נתעלם מהן? זכור כי כל התגיות שלא נמצאת להן התאמה מסווגות כ-<unknown>. קיומן ידוע לנו, אנו פשוט לא רוצים אותן, ולכן אנו מוצאים להן התאמה מפורשת ומתעלמים מהן.

תסריט XSL

מסתבר שכש-Word מייצא את הקובץ ל-HTML, הוא יוצר מספר תגיות Span שונות. בדיקה מהירה של פרק 3 שלנו בקובץ ה-XHTML מגלה כמה כאלו:

```
<span style="mso-tab-count: 1">
<span style="mso-spacerun: yes">
```

אנו נמצא גם שורות עם "style="mso-bookmark ו-"style="font-family או "style="font-style כמו גם "style="layout-grid-mode. אנו מעוניינים לנקוט בפעולות שונות עבור כל אחת מאלו, והדרך הקלה ביותר לעשות זאת היא על ידי ניתוח תחבירי של כל סגנון שנמצא בתגיות ה-span, ופעולה מתאימה, על פי סגנון זה.

בשורה 64 אנו מבצעים התאמה לכל רכיב מטיפוס Span שיש לו את התכונה style:

```
64:              that have a style attribute -->
```

```
65:   <xsl:template match="span[@style]">
```

התאמת התבניות מתבצעת רק עד לכאן. כדי לצמצם לסגנון המתאים והמדויק, אנו פונים לעזרת תסריט (Script).

ב- ++C או Java היינו יוצרים הצהרת switch (הסתעפות); ב-Visual Basic היינו יוצרים הצהרת Select Case. ב-XSL אנו רושמים xsl:choose:

```
66:        <xsl:choose>
```

בדיוק כמו שהצהרת switch ב- ++C או Java מלווה בהצהרת case (מקרה), והצהרת Select ב-VB מלווה בהצהרת Case, כך גם הצהרת choose ב-XSL מלווה בסדרת הצהרות של xsl:when. התנאי הראשון שאתו נעבוד הוא כשהסגנון הוא טאב:

```
68:   <xsl:when expr="getStyleName(this) == 'mso-tab-count'">
69:         <tab />
70:   </xsl:when>
```

בשורה 68 חלק ה-when יופעל כאשר ערך הביטוי הוא true. הביטוי הוא "'getStyleName(this) == 'mso-tab-count".

Expr הוא מבחן בוליאני עבור הצהרת ה-xsl:when. אם המבחן מחזיר true, התוכן של when (במקרה זה מוצג בשורה 69) יושם בפלט. התוצאה של expr בעל ערך true, במקרה זה, שיועבר טאב לפלט. שים לב כי הטאב משתמש בתגית סגירה הסוגרת את עצמה.

GetStyleName אינו מבחן XSL סטנדרטי. זוהי פונקציה שכתבנו ב-xsl:script. אנו מעבירים את הרכיב הנוכחי כפרמטר. this היא מילה שמורה ב-XSL אשר מסמלת את הרכיב הנוכחי.

שלא כמו HTML, ל-XML אין יכולת להוסיף תסריטים בגוף המסמך, אך ל-XSL יש. תסריט ב-XSL מתחילה בתגית xsl:script, כמוצג בשורה 108. בהמשך מצהירים על חלק CDATA. כפי שאמרנו קודם, CDATA הוא טקסט שלא אמור להיות מנותח תחבירית על ידי מנתח תחביר ה-XSL. אנו עושים זאת מכיון שהתסריט עשוי לכלול סמלים כמו < or >, השמורים על ידי XSL. על ידי כתיבת התסריט בתוך קטע CDATA, מנתח התחביר לא ינסה לתרגם את הסמלים האלה ל-XSL.

קטעי CDATA מתחילים ברצף התווים <![CDATA[(כפי שניתן לראות בשורה 109) ומסתיימים ברצף]]> (כפי שניתן לראות בשורה 128):

```
122:  function getStyleName(me)
123:  {
124:        var styleArg = me.getAttribute("style");
```

```
125:        var p = styleArg.indexOf(':');
126:        return styleArg.substr(0, p);
127:  }
```

הרכיב מועבר כפרמטר. בשורה 124 אנו מקבלים את התכונה מהרכיב על ידי קריאה ל-getAttribute, שיטה קיימת לכל אובייקטי DOM ב-XSL; תוך העברת שם התכונה שאנו מעוניינים לקבל. אנו מאחסנים זאת במשתנה מקומי הקרוי styleArg. אחר כך אנו מאתרים את הנקודתיים (:) בתכונה, ומחזירים את כל מה שלפני הנקודתיים. במקרה שבו נמצאה התאמה ל:

```
<span style="mso-tab-count: 1">
```

יוחזר:

```
mso-tab-count
```

אנו משווים ערך מוחזר זה בשורה 68. במקרה שאכן יש לנו mso-tab-count, ערכו של expr יהפוך ל-true, והטאב ייכתב לקובץ הפלט. אנו סוגרים את התגית xsl:when שפתחנו בשורה 68 על ידי תגית סגירה בשורה 70.

בשורות 74-76 אנו מטפלים במקרה בו ישנה התאמה ל-mso-spacerun. הלוגיקה של התאמת הסגנון זהה, אך הפעולה שאנו נוקטים מעט שונה.

אם ישנה התאמה ל-spacerun בשורה 4, אנו פולטים תגית <spacerun> בשורה 75. אחר כך אנו משייכים תכונה לתגית זו תוך שימוש ב-<xsl:attribute name="len">. כפי שצוין קודם לכן, זה יוצר בפלט תגית:

```
<spacerun len=
```

כעת אנו צריכים למלא ערך עבור התכונה len. כפי שניתן לזכור, xsl:attribute **ימלא** כערך של התכונה החדשה את מה שבין התגית xsl:attribute ותגית הסגירה שלה. במקרה שלנו, מה שנמצא בין התגיות האלו הוא:

```
<xsl:eval>CountOccurences(this.firstChild.nodeValue, '\xa0')</xsl:eval>
```

התגית xsl:eval **תעריך ותחזיר** את הערך של כל מה שבין תגית הפתיחה והסגירה שלה. במקרה שלנו, מה שיש בין תגיות אלו זה קריאה לפונקציית התסריט שלנו CountOccurences, אליה אנו מעבירים שני פרמטרים:

```
this.firstChild.nodeValue, '\xa0'
```

הפרמטר הראשון הוא ערך הצומת של הבן הראשון של הרכיב הנוכחי. זכרו כי הרכיב הנוכחי במקרה זה הוא Span. הבן הראשון יהיה רכיב טקסט, שערכו הוא הטקסט עצמו. לכן, קיבלנו את כל מה שמוחזק ב-span. מסתבר שכש-Word יוצר את תגיות ה-span האלו, המייצגות מרווחים רגילים, הוא שם תו רווח בודד ואז סדרה של תווי a0 הקסדצימליים (בסיס 16): אחד לכל תו רווח שהוא מכיל. אנו פשוט מעוניינים לספור את תווי a0 הללו. בפונקציה הבאה אנו סופרים אותם, CountOccurences מוצגת בשורות 111-113:

```
111:  function CountOccurrences(s, c)
112:  {
113:        var n = 0;
```

```
114:         for(var i = 0; i < s.length; i++)
115:         {
116:                if (s.charAt(i) == c) n++;
117:         }
118:         return n;
119:  }
```

המחרוזת מיוצגת על ידי הפרמטר s; התו על ידי הפרמטר c. הלולאה for מגדילה את n בכל פעם שנמצא מופע של c במחרוזת s, ומחזירה ערך זה.

התוצאה בפועל של קריאה זו ל-xsl:eval בשורה 75 היא לשיוך הערך, מספר המופעים של a0, לתכונה len של התגית spacerun החדשה. שאר הכתוב בשורה 75 פשוט סוגר את התגיות eval, attributes ו-spacerun.

בשורות 79-81 אנו מבצעים התאמה לתגיות mso-bookmarks, ומתעלמים מהן. אנו כן קוראים ל-apply-templates כך שאם ישנם רכיבים בתוך הסימניה (Bookmark), או הטקסט, נאסוף אותם עם תבניות ההתאמה האחרות שלנו:

```
79:         <xsl:when expr="getStyleName(this) == 'mso-bookmark'">
80:                <xsl:apply-templates />
81:         </xsl:when>
```

אותה הלוגיקה ננקטת במקרה של font-family, font-style ו-layout-grid-mode, מכולן מתעלמים כפי שניתן לראות בשורות 83-95:

```
83:                <!-- we'll also ignore local font changes -->
84:                <xsl:when expr="getStyleName(this) == 'font-family'">
85:                       <xsl:apply-templates />
86:                </xsl:when>
87:
88:                <xsl:when expr="getStyleName(this) == 'font-style'">
89:                       <xsl:apply-templates />
90:                </xsl:when>
91:
92:                <!-- can't even figure out what this is, so we'll ignore it -->
93:                <xsl:when expr="getStyleName(this) == 'layout-grid-mode'">
94:                       <xsl:apply-templates />
95:                </xsl:when>
```

לבסוף, אנו חייבים להבטיח שאין סגנונות span שחמקו מאיתנו מבלי ששמנו לכך לב, ולכן אנו יוצרים להם מלכוד בדומה למלכוד שיצרנו לשמות תגיות HTML לא ידועים.

בשורה 98 אנו קוראים ל-xsl:otherwise. בדיוק כמו שב- C++ וב-Java יש ברירת מחדל בהצהרות ה-switch שלהם, ול-VB יש את ברירת המחדל else בהצהרת ה-select-case שלו, המסמלת "כל מקרה אחר". בדיוק כך, ב-XSL יש: otherwise. אם לא נמצאה

התאמה לכל תבנית -mso אחרת, ה-otherwise יתאים. במקרה זה, בשורה 99 אנו יוצרים תגית <unknown> עם התכונה tag, שערכה הוא span. אחר כך אנו משייכים תכונה **שנייה** לתגית הלא ידועה שלנו, הקרויה style, שערכה הוא סגנון -mso:

```
98:          <xsl:otherwise>
99:               <unknown>
100:                   <xsl:attribute name="tag">span</xsl:attribute>
101:                   <xsl:attribute name="val"><xsl:eval>
                       ➥getStyleName(this)</xsl:eval></xsl:attribute>
102:                   <xsl:apply-templates />
103:              </unknown>
```

לבסוף, בשורות 104 ו- 105 אנו סוגרים את ההצהרות otherwise ו-choose, ואחר כך את הצהרת ה-template בשורה 106:

```
104:          </xsl:otherwise>
105:     </xsl:choose>
106: </xsl:template>
```

הרצת XHTML ל-XML

עקבנו אחרי control.asp עד לשורה 130, עד לנקודה בה קראנו ל-transform. תוך כדי כך, חקרנו את נושא גיליונות ה-XSL והטרנספורמציה. בתום תהליך הטרנספורמציה, אנו חוזרים ל-control.asp, כמוצג בתדפיס 4.8.

תדפיס 4.8

```
120: Function XHTML2XML()
121:      'convert XHTML file to our canonical XML form
122:      dim oXSL, oH2X, xmlFN, fn0, dtdFN
123:
124:      set oXSL = Server.CreateObject("FromScratch.XSLTransform")
125:
126:      'convert xhtml to xml via a stylesheet
127:      oXSL.InputFile = dataPath & ".xhtml"
128:      oXSL.XSLFile = fso.BuildPath(codeDir, "WordXHTML2XML.xsl")
129:
130:      oXSL.Transform
131:
132:      'save output as .xml0
133:      xmlFN = dataPath & ".xml"
134:      fn0 = xmlFN & "0"
135:      oXSL.SaveOutputAsFile fn0
136:
137:      'we assume that the dtd is in the same directory as the data
138:      dtdFN = "canon.dtd"
```

```
139:
140:        'now pipe the .xml0 to the VB component to do the rest
            ➥of the transformation
141:        set oH2X = Server.CreateObject("FromScratch.WordXHTML2XML")
142:        oH2X.Convert fn0, xmlFN, dtdFN
143:
144:        'and we no longer need the intermediate file
145:        fso.DeleteFile fn0
146:
147:        XHTML2XML = "Converted " & dataPath & ".xhtml" & " to " & xmlFN
148:  End Function
```

הפלט של מעבר ה-XSL מאוחסן בקובץ XML עם המספר 0 המשורשר בסוף. כך, שאם ביצענו את המעבר על Chap2, קובץ הפלט הזמני יהיה Chap2.xml0.

קובץ זה, Chap2.xml0, משמש כקלט לשלב הבא בטרנספורמציה, המובא בפרק הבא. עם השלמת השלב הבא, הקובץ הזמני נמחק, כמוצג בשורה 145.

התוכנית לא מפסיקה בין השלב הראשון לשלב השני. השלבים מתבצעים ברצף באופן אוטומטי; בזמן שהתוכנית מדווחת על סיומה (והדבר קורה די מהר), הקובץ הזמני כבר יימחק. במידה ונרצה לבחון את הקובץ הזמני, נוכל לעשות זאת על ידי הפיכת שורה 145 להערה, ובכך למנוע את מחיקת הקובץ:

```
145:        fso.DeleteFile fn0
```

השוואת הקובץ הזמני עם קובץ ה-XML הסופי יכולה לסייע לך להבין איזו עבודה מבוצעת על ידי טרנספורמצית ה-XSL, ואיזו עבודה מבוצעת על ידי שינוי ה-DOM שב-XML, המובא בהמשך.

הצעדים הבאים

סיימנו שלב ראשון במעבר מקובץ ה-XHTML למסמך XML. בפרק הבא נמשיך בשלב השני והאחרון של המעבר ל-canoncial form שלנו. כדי להשלים את המעבר הזה אנחנו ניצור שינויים ישירות ב-DOM של מסמך ה-XML.

פרק 5
ביצוע שינויים ב-DOM

בפרק זה:

- בחינת קובץ הביניים
- יצירת קטעים
- המעבר מ-XHTML ל-XML - חלק II
- איסוף קטעים: אסטרטגיה
- יצירת ההיררכיה
- בלוקי קוד
- הצעדים הבאים

לאחר הטרנספורמציות שביצענו בפרק 4, יש בידינו קובץ זמני, בדרך אל קובץ ה-XML הסופי שלנו העומד ב-canoncial form שלנו. עד כה השתמשנו ב-XSL לשינוי התגיות; כעת נותר לנו ליצור קיבוץ של הקטעים לכדי מבנה הירכי, ולצרף את רשימות הקוד וההערות.

אנו נבצע את המטלות הללו בשני שלבים. ראשית ניצור את ההיררכיה של הקטעים על ידי סריקת ה-DOM של מסמך ה-XML הקיים ויצירת DOM חדש לפלט. ה-DOM החדש ישקף את ההיררכיה של הקטעים.

שנית, נבצע שינויים ב-DOM החדש הזה כדי ליצור קיבוץ של רשימות הקוד וההערות. שינוי הרכיבים יעשה ב-DOM עצמו. זאת במקום לפלוט את המבנה החדש לפלט DOM נוסף.

אם ברצונך לארגן מחדש את מדף הספרים שלך, ישנן שתי דרכים בהן תוכל לעשות זאת. בדרך הראשונה, תבנה מדף ספרים חדש, ואז תיקח ספרים מהמדף הישן ותמקם אותם היכן שאתה רוצה על המדף החדש. הדרך השנייה היא להסיר ספרים מהמדף ולהכניס אותם חזרה לאותו המדף, אך בסדר אחר. כלומר, לשנות את הסדר בתוך המדף עצמו. בתכנות זה דומה מאוד לשינויים על מערכים (למשל, מיון מערך) - אתה יכול לשנות את המערך על ידי שינוי סדר האיברים שבתוכו, ואתה יכול גם לעשות זאת על ידי שימוש במערך עזר.

בחינת קובץ הביניים

קבצי הביניים שנקבל לאחר פעולת XSLTransform קרויים chap?.xml0. למשל, קובץ הביניים עבור פרק 3 הוא Chap3.xml0. הבה נבחן קטע מתוך קובץ זה, ונקבל מושג לגבי הצורך בעבודה נוספת, כפי שניתן לראות בתדפיס 5.1.

תדפיס 5.1

```
<?xml version="1.0"?>
<html>
  <div class="HA">
(a)3
      </div>
  <div class="HB">
(b)Proof of concept
      </div>
  <div class="FT">
Enough theory! Before we go any further in thinking through how we'll implement
   ➥EmployeeNet, we need to take a look at the implementation technology and get
   ➥something working.
      </div>
      <div class="PD">
***Begin Note***
      </div>
 <noteline>
Here are the steps for Visual InterDev. As explained in the previous chapter,
   ➥the exact details may be different on your computer or network, or if you are
   ➥using different Internet enabling technology.
      </noteline>
 <div class="PD">
***End Note***
Listing 3.2
      </div>
 <codeline>
1: Function GetTheData() As Recordset
      </codeline>
 <codeline>
2:
         <spacerun len="4"/>

Dim rs As ADODB.Recordset
      </codeline>
 <codeline>
3:
         <spacerun len="4"/>

```

```
997:  Set rs = New Recordset
998:        </codeline>
999:  <codeline>
1000:  4:
1001:          <spacerun len="4"/>
1002:
1003:  Call rs.Open("select * from publishers", "dsn=pubs",
1004:          <spacerun len="1"/>
1005:
1006:  UID=sa; PWD=;")
1007:             </codeline>
1008:         <codeline>
1009:  5:
1010:          <spacerun len="4"/>
1011:
1012:  Set GetTheData = rs
1013:            </codeline>
1014:         <codeline>
1015:  6: End Function
1016:           </codeline>
1017:         <div class="FT">
1018:  Your screen should look like Figure 3.11, with the project listed in the project
      ➥window on the right, and the code shown in the GetTheData method on the left.
1019:        </div>
```

השמטתי חלק ניכר מהתדפיס כדי לחסוך במקום, והתמקדתי בשלושה תחומי עניין מרכזיים. בעוד שקובץ זה קרוב יותר ל-canoncial form שלנו מאשר קובץ ה-XHTML, הוא עדיין לא עומד בדרישות המוצגות ב-DTD שלנו, ואינו חוקי.

ניתוח

בשורה 1 אנו רואים את הרכיב xml המציין שקובץ זה הוא XML בגירסה 1. יש לזכור כי בעוד שקובץ זה אינו חוקי (מכיוון שהוא לא עומד בדרישות ה-DTD שלנו) הוא עדיין מסמך XML בנוי היטב (עומד בדרישות המפרט של XML).

שורה 2 מראה כי רכיב השורש הוא <html>. אבל, ב-DTD שלנו קבענו את רכיב השורש כ- <book>. נתקן זאת בטרנספורמציה הבאה שלנו.

נשים לב גם כי אין התייחסות ל-DTD שלנו בקובץ הביניים. לכן, יש להוסיף לקובץ ה-XML הסופי את השורה הבאה:

```
1:      <!DOCTYPE book SYSTEM "canon.dtd">
```

הוספת השורה הזו תתבצע בטרנספורמציה ממסמך הביניים למסמך הסופי.
הרכיב DOCTYPE הוא אופציונלי בכל מסמך XML. אם הוא אכן מופיע, הוא יכול להופיע רק פעם אחת, והוא מגדיר מהו ה-DTD עבור דף זה. לאחר מילת המפתח

DOCTYPE נכתב שם המסמך (במקרה שלנו book) ולאחריו ייכתב הרכיב PUBLIC או SYSTEM.

מילת המפתח PUBLIC באה לאחר המזהה (Identifier), בו ייעשה שימוש על ידי מנתח התחביר כדי לקבל את ה-DTD מתוך מאגר פנימי או חיצוני. בדרך כלל הוא בא יחד עם URL, כך שאם מנתח התחביר לא יכול למצוא את ה-DTD באמצעות המזהה, הוא יוכל להשתמש ב-URL.

במקרה שלנו נשתמש במילת המפתח SYSTEM, המורה למנתח התחביר ללכת ישירות ל-URL המסופק. אנו מספקים גם שם של קובץ, שימצא באותה תיקייה בה מצוי מסמך המקור.

בחינת קובץ הביניים בפירוט

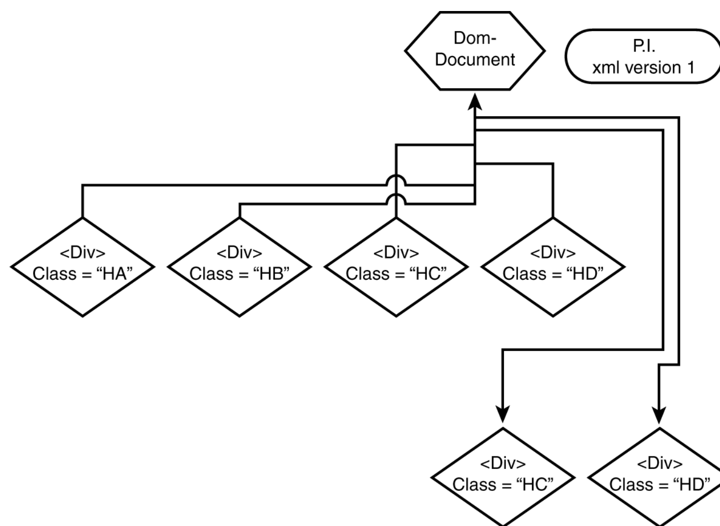
שורות 3-7 של תדפיס 5.1 מראות כי שינינו את הכותרת ברמה-A שלנו ל- <div>, וזה רק לטובה, אך בדרישות ה-DTD החלטנו שכותרות אלו ימוקמו ב- <section>, ועדיין לא השגנו זאת.

בהמשך קובץ הביניים, בשורה 6 אנו רואים שהכותרת ברמה-B הוכרזה כ-div. שוב, עדיין לא קיים כל יחס בין הקטעים (sections) בקובץ ה-XML; הם ממשיכים לשקף את התצורה הליניארית של המסמך המקורי. בטרנספורמציה הבאה, המטרה שלנו תהיה ליצור מבנה היררכי של יחסים, בין הקטעים השונים.

שורות 173-175 מראות את תגיות שורת ההערה (Noteline) שיצרנו בטרנספורמציית ה-XSL, אך ע״פ דרישות ה-DTD שלנו, אלו אמורות להיות מוכלות בין התגיות <note> ו- </note>. באופן דומה, שורות 984-1016 מראות סדרה של תגיות שורת קוד (Codeline), אך שוב, ה-DTD דורש שאלו תהיינה תחומות על ידי התגיות <code> ו- </code>. את כל הבעיות הללו נתקן בסבב הזה של טרנספורמציות.

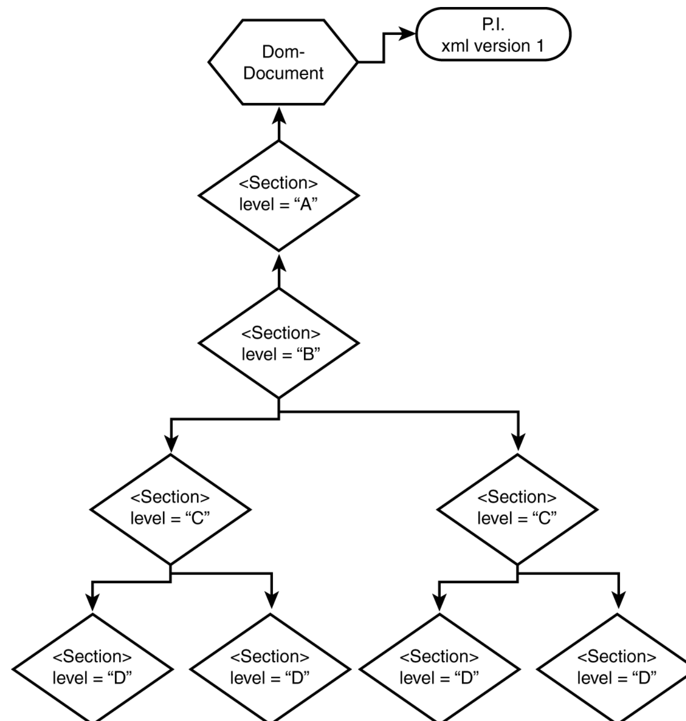
יצירת קטעים (sections)

המשימה שלנו היא להעביר את הכותרות ברמה-A, -B, -C, ו- -D למבנה יחסים היררכי בין קטעים. אם היינו צריכים לקרוא את קובץ הביניים לתוך מסמך ה-DOM הוא היה מאוד שטוח (flat). כל הכותרות ברמה A-D היו מיידית נמצאות תחת רכיב השורש, כפי שניתן לראות בתרשים 5.1.



תרשים 5.1: מסמך הביניים שטוח

במקום זאת אנו רוצים להפוך אותן לקטעים המאורגנים בהיררכיה, כמוצג בתרשים 5.2.



תרשים 5.2: ארגון היררכי

כפי שניתן לראות, מסמך ה-DOM החדש ישקף את היחסים בין הקטעים כפי שהיינו חושבים עליהם באופן טבעי: קטעי C מכילים קטעי D, ומוכלים בתוך קטעי B.

ההגדרה של קטע

הבה נביט בהגדרת ה-DTD של רכיב הקטע section. תדפיס 5.2 הוא חלק מה-DTD כפי שהופיע בתדפיס 3.9 בפרק 3 :

תדפיס 5.2

```
<!- - Sections hvae level and id attributes and begin with a title
sections may contain other sections - - >
<!ELEMENT section (title, (section | div | note | code)*)>
<!ATTLIST section
      level CDATA #REQUIRED
      id CDATA #REQUIRED>
```

החלטנו שלכל קטע תהיה כותרת אחת בדיוק (כלומר, חייבת להיות לו כותרת אחת ורק אחת). כמו כן, יכולים להיות לו אפס או יותר קטעים אחרים של בלוקי code, note, או div בתוכו. בנוסף לכך, ניתן לראות כי לכל רכיב שתי תכונות **דרושות** : level ו-ID, שניהם מטיפוס CDATA – שהוא טקסט שאינו מנותח תחבירית.

סיור קצר

תזכורת: PCData הוא טקסט המנותח תחבירית: כלומר, זהו טקסט פשוט שמנתח תחביר ה-XML ישתמש בו לבחינת תגיות כך שהוא עשוי להשתנות בהתאם. CDATA הוא טקסט שאינו לניתוח תחבירי: אנו משתמשים בו בכל פעם שעשויים להיות לנו סמלים מיוחדים כמו הסימן קטן מ- (<) או גדול מ- (>), אשר יבלבלו את מנתח התחביר.

תדפיס 5.3 ממחיש כיצד הכותרות ברמה-A ו-B ייראו בקובץ ה-XML הסופי שלנו, כמו גם איך התגיות <note> תתוספנה.

תדפיס 5.3

```
<section level="A" ID="3000">
<title>Chapter 3</title>
<section level="B" ID="3001">
<title>Proof of concept</title>

<note>
      <noteline>
            Here are the steps for Visual InterDev. As Explained in the previous chapter,
            ➥the exact details may be different on your computer or network, or if you
            ➥are using different Internet enabling technology.
            </noteline>
</note>
```

המעבר מ-XHTML ל-XML – חלק II

נזכור כי הקובץ control.asp שלנו יצר את הקובץ (הזמני) html0. הוא גם ישגר את השלב השני בטרנספורמציה שלנו. תדפיס 5.4 מציג את הקטע הרלוונטי מה-Control.asp.

תדפיס 5.4

```
120: Function XHTML2XML()
121:     'convert XHTML file to our canonical XML form
122:     dim oXSL, oH2X, xmlFN, fn0, dtdFN
123:
124:     set oXSL = Server.CreateObject("FromScratch.XSLTransform")
125:
126:     'convert xhtml to xml via a stylesheet
127:     oXSL.InputFile = dataPath & ".xhtml"
128:     oXSL.XSLFile = fso.BuildPath(codeDir, "WordXHTML2XML.xsl")
129:
130:     oXSL.Transform
131:
132:     'save output as .xml0
133:     xmlFN = dataPath & ".xml"
134:     fn0 = xmlFN & "0"
135:     oXSL.SaveOutputAsFile fn0
136:
137:     'we assume that the dtd is in the same directory as the data
138:     dtdFN = "canon.dtd"
139:
140:     'now pipe the .xml0 to the VB component to do the rest
         ➥of the transformation
141:     set oH2X = Server.CreateObject("FromScratch.WordXHTML2XML")
142:         oH2X.Convert fn0, xmlFN, dtdFN
143:
144:         'and we no longer need the intermediate file
145:         fso.DeleteFile fn0
146:
147:     XHTML2XML = "Converted " & dataPath & ".xhtml" & " to " & xmlFN
148: End Function
```

ניתוח

שורות 120-135 מבססות את הידע שהוצג בפרק 4. קובץ הפלט (במקרה זה Chap3.xml0) נשמר בשורה 135.

בשורה 138 המשתנה הלוקאלי dtdFN מקבל את הערך "canon.dtd" – שהוא שם קובץ ה-DTD שלנו.



כדי לשמור על קוד זה פשוט, אנו מניחים שה-canon.dtd הוא באותה התיקיה של קבצי המידע. ניתן להפוך את הקוד לגמיש יותר על ידי שימוש ב-URL כדי להצביע על ה-DTD. אם רוצים להשתמש בזה כפי שהצגנו, זיכרו כי יש להעתיק את הקובץ canon.dtd (מצוי בתקליטור) לתיקיה המכילה את chap3.xml0.

בשורה 141 אנו יוצרים מופע של אובייקט ה-WordXHTML2XML ActiveX ובשורה 142 אנו קוראים לשיטה Convert על אובייקט זה, תוך העברת קובץ הביניים (Chap3.xml0), קובץ הפלט (chap3.xml) וקובץ ה-dtd (canon.dtd).

מודול זה כתבנו ב-Visual Basic, כמוצג בתדפיס 5.5.

תדפיס 5.5

```
'Takes the result of the first phase of XHTML to XML conversion (performed by the
'stylesheet) and then continues the conversion process, doing the "bundling"
'of elements.
Public Sub Convert(ByVal inPath As String, ByVal outPath As String,
 ➥Optional ByVal dtdPath As String = "")
   Dim inDom As New DOMDocument, outDom As DOMDocument, outStr As String,
   ➥dtdStr As String

   'load the XML from the file into a DOM
   inDom.async = False
   inDom.Load inPath

   'the first operation is collecting the A,B,C,... level sections
   Set outDom = CollectSections(inDom)

   'next we collect all the contiguous <codeline> lines into a <code> tag
   CollectContig outDom, "code"

   'ditto for note lines
   CollectContig outDom, "note"

   'get the results into a string
   outStr = outDom.xml

   'if a DTD has been specified, insert the doctype and external reference
   'seems like there should be a way to do it via the DOM, but I can't figure it out
   If dtdPath <> "" Then
       dtdStr = "<!DOCTYPE book SYSTEM """ & dtdPath & """>" & vbCrLf
       'we want to insert it right after the xml pi
       outStr = Replace(outStr, "?>" & vbCrLf, "?>" & vbCrLf & dtdStr, , 1)
   End If
```

```
29:
30:     'and now we save the results into a file
31:     Open outPath For Output As #1
32:     Print #1, outStr
33:     Close #1
34: End Sub
```

ניתוח

בשורה 3 ניתן לראות את הפרמטרים המועברים מקובץ ה-asp השולט. בשורה 4 אנו יוצרים ארבעה משתנים מקומיים. השניים הראשונים הם אובייקטי מסמך DOM.

אובייקט ה-XML היחידי שנוצר ישירות הוא Dom Document. כל שאר האובייקטים, כמו הרכיבים למשל, נוצרים באמצעות שיטות של DomDocument, כפי שנראה מייד. בשורה 7 אנו קובעים את המאפיין async של DOM הקלט שלנו ל-false. משמעות הדבר היא שכשנטען את המסמך הזה, התוכנית שלנו תחכה עד שהמסמך ייטען במלואו לפני שתמשיך. טעינה אסינכרונית נוחה מאוד כאשר התוכנית עלולה להיות עסוקה במטלות אחרות, אך במקרה זה אנו מעדיפים לחכות.

בשורה 8 אנו טוענים את המסמך, תוך העברת שם הקובץ לטעינה. היחס בין קובץ זה (Chap3.xml0) וה-DOM שבזיכרון הוא יחס עדין יותר ממה שנראה במבט ראשון.

הגישה מונחית העצמים

כל מתכנת מבין את ההבדל בין קובץ על דיסק ובין הייצוג של אותו קובץ בזיכרון, אך אנו עדיין נלכדים בדקויות שבהתקשרות עם ה-DOM. להלן הסיבה לכך: אנו למעשה מטפלים בארבע יישויות, לא בשתיים.

- יישות 1: המסמך הפיזי בדיסק: סדרת בתים המאוחסנת על ידי מערכת ההפעלה באחסון קבוע.
- יישות 2: המסמך הלוגי כפי שהיינו עשויים לראות אותו בעורך או בדפדפן.
- יישות 3: ה-DOM הפיזי – בתים בזיכרון.
- יישות 4: ה-DOM הלוגי – הרכיבים והצמתים בהירככיה כמתואר בהמלצת W3C.

הבה נהיה ברורים, עד סוף הספר אני לא אדבר על הבתים הפיזיים, לא בדיסק ולא בזיכרון. אני מעוניין רק ביישויות הלוגיות.



כן, כמובן שבסופו של דבר המסמך וה-DOM צריכים לבוא לידי ביטוי בבתים, אך למה לעצור שם? בתים הם למעשה רק תופעת לוואי המתקבלת ממצב רגעי של מעגלים חשמליים. למעשה, מעגלים חשמליים הם רק מטאפורה למצב כמות החלקיקים הלא-דטרמיניסטיים. הדבר שמעניין אותנו הוא תמיד ההפשטה הלוגית; הבתים הם פשוט מנגנון יישום.

המסמך וה-DOM

היחס בין המסמך וה-DOM הוא מורכב ומעניין. ה-DOM (Document Object Model) הינו ייצוג מונחה עצמים של המסמך. כל רכיב במסמך מיוצג על ידי אובייקט ב-DOM. היחס הוא הדדי: המסמך הוא ייצוג ליניארי של ה-DOM.

כשאנו יוצרים מסמך DOM (מופע של DOMDOcument) אנו יכולים לאכלס אותו על ידי הוספת רכיבים בעת הצורך, או על ידי קריאה לשיטה Load של מסמך ה-DOM החדש והעברת מסמך XML. במקרה האחרון, ה-DOM המתקבל יהיה מובנה לחלוטין; בייצוג אובייקטים ישיר של מסמך ה-XML שהועבר. זהו האופן בו אנו יוצרים את מסמך DOM הקלט שלנו, כמוצג בשורה 8:

```
8:    inDom.Load inPath
```

בשורה 9 אנו קוראים ל-CollectSections, תוך העברת DOM הקלט החדש שנטען במלואו. הערך המוחזר מהקריאה לשיטה יהיה מסמך DOM חדש, שנשייך אותו ל-outDom.

איסוף קטעים: אסטרטגיה

נזכור כי בתרשים 5.1 הראינו עץ שטוח, עם כותרות ברמה-A, -B, -C ו-D, שכולן מקושרות לשורש. האסטרטגיה שלנו תהיה לסרוק את העץ, וליצור עץ חדש, עם הקטעים החדשים שנוצרו, במיקום הנכון.

יש לזכור כי ישנם גם רכיבים רבים אחרים בעץ המקורי. למעשה, תרשים 5.1 יהיה יותר מדויק (אך עדיין מפושט יחסית) אם יכלול רכיבים כמו טקסט, הערות, קוד וכן הלאה. האסטרטגיה שלנו תהיה לסרוק את DOM המקור (הנוצר מקובץ הביניים), ולבחון כל רכיב. אם זהו רכיב <DIV> עם "class="H?, ניצור רכיב <section> חדש ב-DOM היעד (שמאוחר יותר נשמור כקובץ ה-xml. שלנו).

נשתמש במשתנה מקומי הקרוי curParent, שבתחילה יצביע על רכיב השורש. נוסיף את רכיב הקטע section הראשון שלנו כרכיב בן של curParent (שורש), ואז נכוון את curParent להצביע על קטע זה.

נמשיך לסרוק את מסמך DOM המקור, ונוסיף כל רכיב שנמצא כילד של curParent (הקטע הראשון שלנו) עד שניתקל ברכיב <DIV> עם "class="H?. בנקודה זו ניצור רכיב קטע section חדש, נהפוך אותו לרכיב בן של curParent, ונכוון את curParent להצביע על הקטע החדש שזה עתה הוספנו.

שוב, נסרוק את העץ, כשהפעם נהפוך כל רכיב שנמצא לבן של curParent הנוכחי (הבן השני שלנו).

נחזור על תהליך זה שוב ושוב עד שנעביר את כל הרכיבים שב-DOM המקור שלנו ל-DOM היעד.

איסוף קטעים: יישום

ניישם אסטרטגיה זו באמצעות השיטה CollectionSections כפי שניתן לראות בתדפיס 5.6.

תדפיס 5.6

```
'in Word, we just know when a section begins. What we want is the entire contents of
'a given section (including all descendants) to be enclosed in a <section> tag

'the approach is to walk thru all the elements, looking for section headers
'we create a new node to serve as the section element
'as we encounter each new section head, we push or pop the current section element
'based on the "level" of the section

'note that here we walk thru all the input and build up a new output tree (vs. below)
Private Function CollectSections(inDom As DOMDocument) As DOMDocument
    Dim i As Long, outDom As DOMDocument, children As IXMLDOMNodeList
    Dim c As IXMLDOMElement, className As String, curParent As IXMLDOMElement
    Dim newC As IXMLDOMElement, omit As Boolean, title As IXMLDOMElement,
    ➥s As String
    Dim sectionId As Long, pi As IXMLDOMProcessingInstruction

    'we're going to give each section a unique id, but we need to know where to start
    'for these documents, the A head in each chapter is a chapter number,
    ➥so we'll multiply that by 1000
    'and use the result as the starting id
    Set c = inDom.selectSingleNode("//div[@class='HA']")
    sectionId = CLng(StripParenLetter(c.Text) & "000")

    'create a new DOM for the output
    Set outDom = New DOMDocument

    'add the XML processing instruction
    Set pi = outDom.createProcessingInstruction("xml", "version='1.0'")
    outDom.appendChild pi

    'and give it a top level element - which will also be where we begin inserting
    ➥the output
    Set curParent = outDom.createElement("book")
    curParent.setAttribute "level", "0"   'temp
    outDom.appendChild curParent

    'we want to enumerate all the immediate children of the top-level element
    ➥(<html>) of the input
```

```
Set children = inDom.documentElement.childNodes

For i = 0 To children.length - 1
    omit = False
    If children(i).nodeType = NODE_ELEMENT Then
        Set c = children(i)
        If c.nodeName = "div" Then
            className = c.getAttribute("class")
            If Left(className, 1) = "H" Then
                'we have a new header - create an element to hold its sub-tree
                Set newC = outDom.createElement("section")
                newC.setAttribute "level", Right(className, 1)
                newC.setAttribute "id", sectionId
                sectionId = sectionId + 1

                'the current contents of this element will become the title of the section
                Set title = outDom.createElement("title")
                s = Trim(c.Text)

                'most of the headers contain "(x)" at the beginning - we can strip that
                s = StripParenLetter(s)

                'A levels just give the chapter number - let's flesh that out
                If className = "HA" Then s = "Chapter " & s

                'now insert the title element as a child of the section
                title.Text = s
                newC.appendChild title

                'what we do depends on the relative level to our current parent
                If newC.getAttribute("level") > curParent.getAttribute("level") Then
                    'higher level, ie. at a greater depth in the tree, create and push
                    ➥a new level
                    curParent.appendChild newC
                ElseIf newC.getAttribute("level") = curParent.getAttribute("level") Then
                    'this is a peer to the current parent
                    curParent.parentNode.appendChild newC
                Else
                    'we are popping the tree
                    Do
                        Set curParent = curParent.parentNode
                        'go until we find a peer
                        If curParent.getAttribute("level") = newC.getAttribute("level")
                        ➥Then Exit Do
```

```
76:                 Loop
77:                 'and we can insert the new element under our parent
78:                 curParent.parentNode.appendChild newC
79:               End If
80:               'now we want all new addtions to be children of newC
81:               Set curParent = newC
82:               'and we don't need to insert ourself
83:               omit = True
84:             End If
85:           End If
86:         End If
87:
88:         'add this node to the output
89:         If Not omit Then curParent.appendChild c.cloneNode(True)
90:     Next
91:
92:     outDom.documentElement.removeAttribute "level"  'don't need it any more
93:
94:     'return the newly constructed dom
95:     Set CollectSections = outDom
96: End Function
```

ניתוח

בשורה 9 ניתן לראות את מסמך DOM המקור מועבר כפרמטר מסוג DOMDocument, וניתן לראות גם את ההצהרה שהפלט של פונקציה זו יהיה מסמך DOM. מסמך פלט זה נוצר בשורה 22.

הבה נחזור אחורה לשורה 18. כאן המשתנה המקומי c, המוגדר בשורה 12 להיות IXMLDOMElement, מקבל את תוצאת הקריאה ל-selectSingleNode על DOM הקלט. השיטה selectSingleNode מקבלת פרמטר: מחרוזת של תבנית. selectsinglenode מחזירה צומת (node) בודד, המתאים לתבנית.

התבנית היא "//div[@class='HA']", תבנית זו מתחלקת לארבעה חלקים:

- חלק 1: //
- חלק 2: div
- חלק 3: [@class=]
- חלק 4: 'HA'

ישנן שתי דרכים לציון חיפוש אחר צאצאים. לוכסן יחיד (/) מציין "הצאצא הישיר של הצומת הנוכחי", ולוכסן כפול (//) מציין "כל הצאצאים של הצומת הנוכחי". במקרה שלנו, אנו מחפשים אחר כל הצאצאים של הצומת הנוכחי (שורש).

חלק 2 הוא סוג הרכיב אותו אנו מחפשים, במקרה שלנו div. לכן נחפש אחר כל רכיבי div שבמסמך.

חלק 3 מציין שאנו רוצים להתאים רק רכיבי div בעלי תכונה מסוימת (הסימן [@] מציין תכונה), וחלק 4 אומר כי התכונה שאנו מחפשים היא 'HA'.

התוצאה הכוללת היא שאנו מבצעים התאמה לכל תגיות <div> עם התכונה class='HA' ומשייכים אותן למשתנה c. בחינה של מסמך הקלט תמצא רק רכיב אחד כזה.

להלן האופן בו נראה רכיב כזה במסמך הקלט, Chap3.xml0:

```
<div class="HA">
(a)3
</div>
```

כתוצאה מן הפעולה הזו, c הוא רכיב <div> עם תכונה class שערכה הוא HA. ל-c יש בן שהוא רכיב טקסט שערכו הוא 3(a).

יצירת ה-ID של הקטע

אנו ממשיכים עם שורה 19, שגם היא מעט מורכבת:

```
19:    sectionId = CLng(StripParenLetter(c.Text) & "000")
```

כאן אנו מקבלים את המאפיין Text מהרכיב שלנו, שהוא 3(a) ומעבירים אותו ל-StripParenLetter, כמוצג בתדפיס 5.7

תדפיס 5.7

```
Private Function StripParenLetter(ByVal s As String) As String
    'remove leading "(x)"
    If Left(s, 1) = "(" And Mid(s, 3, 1) = ")" Then s = Mid(s, 4)
    StripParenLetter = s
End Function
```

התוצאה היא החזרת המחרוזת "3", מספרו של הפרק. אנו משרשרים את המחרוזת "000" ל-"3" ושולחים מחרוזת משורשרת זו לפונקצית CLng המובניית של VB, אשר הופכת את המחרוזת "3000" למספר 3000. הערך הזה מוחזר ל-sectionID.

כל פרק ימוספר בהתאם. כך, לפרק 4 יהיו קטעים שימוספרו בתחום 4000 עד 4999. אנו לוקחים כנתון שאין פרקים עם יותר מ- 999 קטעים או תתי-פרקים (למעשה, אין פרקים עם יותר מ- 100 קטעים). למרות זאת, אנחנו מגזימים ליתר ביטחון.

יצירת הוראות עיבוד

עתה אנו שבים לשורה 22, היכן שמסמך DOM הפלט נוצר. מסמך פלט זה יהיה ב-XML, וכל מסמך XML דורש הוראת עיבוד בראשית הקובץ, כמו זו שלהלן:

```
<?xml version="1.0"?>
```

משימתנו הראשונה היא ליצור את הוראת העיבוד הזו, זאת אנו עושים בשורה 25. שים לב להצהרה של pi בשורה 13; זהו IXMLDomProcessingInstruction.

הוראות עיבוד נוצרות על ידי קריאה ל-()createProcessingInstruction על האובייקט DOMDocuement, תוך העברת שני פרמטרים: המטרה והמידע. המטרה הופכת לשם הצומת (xml) עבור הוראת העיבוד והמידע הופך לערך הצומת ("version=1.0").

עתה כשיש לנו הוראת עיבוד, אנו משרשרים אותה למסמך DOM הפלט בשורה 26. פעולה זו מוסיפה את הוראת העיבוד לאוסף הבנים של מסמך ה-DOM. ל-DOMDocument, כמו לכל רכיב, יש אוסף של בנים; הבן הראשון באוסף זה יהיה עתה הוראת עיבוד זו.

יצירת ההיררכיה

עכשיו אנו מוכנים להפוך את היחס המרומז בין הקטעים במסמך הקלט ליחסים מפורשים במסמך הפלט. נעשה זאת על ידי יצירת קטע section ברמה-B כרכיב בן של התגית ברמה-A, ואחר כך ניצור קטע ברמה-C כבן של הרמה-B. לבסוף, ניצור קטעים ברמה-D כבנים של הקטעים ברמה-C.

כדי להשיג את כל זה, אנו חייבים לעקוב אחר מיקומנו במסמך. כשאנו מגיעים לקטע ברמה-D אנו חייבים להחליט היכן למקם אותו בהיררכיה החדשה. אם הרכיב הנוכחי הוא קטע ברמה-C, אז רמה-D החדשה תהיה הבן שלו. אם הרכיב הנוכחי הוא ברמה-D, אז רמה-D החדשה היא אח, וחייבת להיות מוכנסת כבן של ההורה מן הרכיב הנוכחי. (אחותך היא הילדה של הוריך).

לבסוף אם הרכיב הנוכחי הוא קטע ברמה-E, אז רמה-D החדשה היא דודתו, וחייבת להיות מוכנסת כבן של הסבא מן הרכיב הנוכחי!

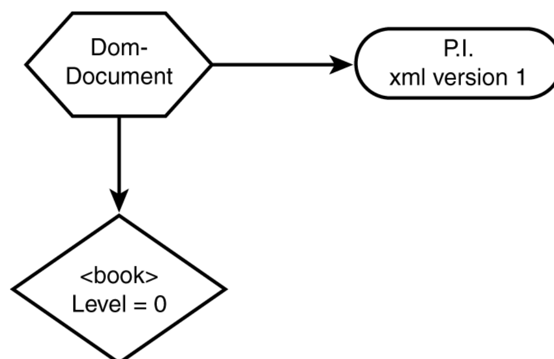
יצירת הרכיב ברמה העליונה

לפי ה-DTD שלנו, הרכיב ברמה העליונה יהיה <book>. אנו מתחילים ביצירת רכיב זה בשורה 29, על ידי קריאה ל- createElement על מסמך DOM הפלט:

```
29:   Set curParent = outDom.createElement("book")
```

createElement מקבלת פרמטר אחד (שם הרכיב) ומחזירה את הרכיב החדש שנוצר. אנו מעבירים את שם הרכיב (book) ומשייכים את הרכיב המתקבל ל- curParent.

כדי לעקוב אחר היחסים, נשייך לכל רכיב תכונה עבור ה"רמה" שלו. רכיב השורש (book) לא יזדקק לציון רמה כשנסיים, אך בזמן יצירת העץ יהיה נוח לקבוע לו את הרמה 0, המציינת רמה עליונה ביותר. בשורה 30 אנו נותנים לו את התכונה level, עם ערך 0. בשורה 31 אנו משרשרים את רכיב השורש למסמך DOM הפלט החדש שלנו. מסמך ה-DOM שלנו נראה עתה כמו בתרשים 5.3.



תרשים 5.3: מסמך ה-DOM לאחר שרשור רכיב השורש.

עכשיו אנו מוכנים לסרוק את מסמך DOM הקלט (המייצג את Chap3.xml0), תוך כדי יצירת רכיבים חדשים במסמך DOM הפלט ככל שנתקדם.

לכל מסמך DOM יש מאפיין documentElement, שהוא נקודת צומת השורש במסמך. בשורה 34 אנו ניגשים לצומת זה, שבמקרה של DOM הקלט הוא HTML :

```
34:    Set children - inDom.documentElement.childNodes
```

אחר כך אנו פונים למאפיין childNodes של אלמנט השורש הזה, המחזיר אוסף של כל הצאצאים הישירים של HTML. מסתבר שבמסמך הקלט, **כל** הקטעים הם צאצאים ישירים של HTML. יש לזכור, למסמך הקלט אין כל מכנה הירכי ; זה בדיוק מה שאנו יוצרים עכשיו.

התוצאה בפועל היא שהאוסף children מכיל את כל הרכיבים וצמתי הטקסט ממסמך הקלט. מספר הרכיבים האלו מוחזר על ידי המאפיין length. אנו משתמשים במספר זה בלולאת for בשורה 36, המאפשרת לנו לעבור באיטרציות רכיב אחר רכיב באוסף :

```
36:    For I = 0 To children.length - 1
```

בשורה 37 אנו מאתחלים את המשתנה המקומי omit ל-false. נחזור למשמעות של משתנה זה בעוד רגע.

האוסף childNodes יכיל שני סוגי צמתים : רכיבים וטקסט. בשורה 38 אנו בודקים את סוג הצומת הנוכחי כדי לראות האם זהו רכיב. אם כן, אנו נכנסים לקוד בשורה 39, שם אנו משייכים צומת זה ל-c, אותו הגדרנו להיות IXMLDOMElement. שוב, אנו המרנו במרומז צומת זה לרכיב :

```
38:    If children(i).nodeType = NODE_ELEMENT Then
39:            Set c = children(i)
```

בשורה 40 אנו בודקים האם הרכיב שבידינו הוא <div>. כל הקטעים מסומנים כ-div, כך שהם הרכיבים היחידים שמעניינים אותנו.

אם אכן יש לנו רכיב div, בשורה 41 אנו שולפים את התכונה class שלו, ומשייכים את הערך למשתנה המחרוזת המקומי className. בשורה 42 אנו בודקים האם האות

הראשונה משמאל של התכונה className היא "H", המציינת שיש לנו כותרת. במידה וכן, אנו סוף סוף מוכנים לבצע מעט עבודה, שכן אנו יודעים שיש לנו כותרת של קטע:

```
40:         If c.nodeName = "div" Then
41:             className = c.getAttribute("class")
42:             If Left(className, 1) = "H" Then
```

בשורה 44 אנו יוצרים רכיב חדש מטיפוס section. בשורה 45 אנו משייכים לרכיב חדש זה תכונה, level, שערכה הוא כל מה שבא אחרי H ברכיב <div> המקורי. כך שאם רכיב הקלט הנוכחי הוא HC אז יצרנו עכשיו רכיב section עם התכונה "level = "C. אנו מוסיפים גם תכונה נוספת, id, לה אנו משייכים את הערך sectionID, כפי שתזכור אתחלנו את פרק 3 ל- 3000. בשורה 47 אנו מגדילים את הערך של sectionID:

```
44:                 Set newC = outDom.createElement("section")
45:                 newC.setAttribute "level", Right(className, 1)
46:                 newC.setAttribute "id", sectionId
47:                 sectionId = sectionId + 1
```

על פי ה-DTD שלנו, לכל קטע חייבת להיות כותרת. אנו יוצרים רכיב חדש מסוג title ומשייכים אותו למשתנה המקומי title מסוג IXMLDOMElement.

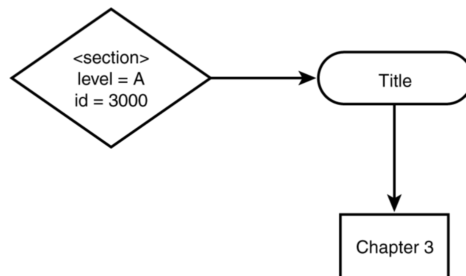
המשימה הבאה שלנו היא לתת ערך לכותרת החדשה שזה עתה יצרנו. נבסס את הכותרת על כותרת מסמך הקלט, אך עם מספר שינויים. אם למשל כותרת הקלט היא "(C) Which Technology", נסיר את ה-(C) ונעביר את הכותרת החדשה "Which Technology".

החריגה היחידה מכך היא הכותרת ברמה-A, אשר בקובץ הקלט אין לה דבר מלבד מספר פרק, למשל, 3(A). אנו נטפל בכך במסמך הפלט שלנו, על ידי הסרת ה-(A) והוספת המילה "Chapter" כדי לקבל Chapter 3.

בשורה 51 אנו קוצצים את כותרת הקלט, על ידי הסרת כל הרווחים שבתחילת ובסוף הכותרת. בשורה 54 אנו מעבירים את המחרוזת ל-StringParenLetter, המסירה את (A). בשורה 54 אנו משליכים את כותרת הפרק, ובשורה 60 אנו קובעים את טקסט הכותרת החדש, למחרוזת שזה עתה בנינו:

```
50:                 Set title = outDom.createElement("title")
51:                 s = Trim(c.Text)
52:
53:                 'most of the headers contain "(x)" at the beginning - we can strip that
54:                 s = StripParenLetter(s)
55:
56:                 'A levels just give the chapter number - let's flesh that out
57:                 If className = "HA" Then s = "Chapter " & s
58:
59:                 'now insert the title element as a child of the section
60:                 title.Text = s
```

הכותרת מוכנה עכשיו ואנו משרשרים אותה ל-newC. כעת יש לנו <section>, המוכן להוספה למסמך DOM הפלט. במקרה ראשון זה, רכיב הקטע section החדש נראה כמו בתרשים 5.4.

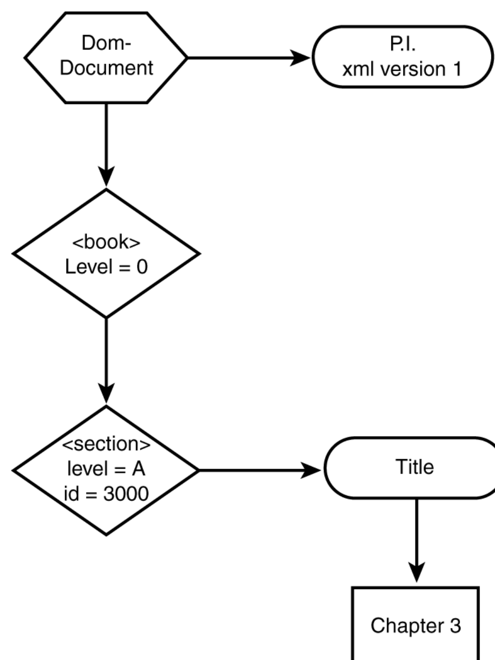


תרשים 5.4: רכיב הקטע החדש

אך היכן אנו מכניסים אותו? בשורה 64 אנו משיגים את התכונה level מהקטע החדש שלנו ומשווים אותה עם הרמה של ההורה הנוכחי:

```
64:         If newC.getAttribute("level") > curParent.getAttribute("level") Then
```

כאן ניתן לראות מדוע סיפקנו רמה (זמנית) לצומת השורש <book>. אנו משווים את רמת הרכיב החדש (A) עם רמת רכיב השורש (0) ומגלים שרכיב חדש זה צריך להיות בנו של השורש. לכן, משפט התנאי שבשורה 64 הוא אמת - true ואנו משרשרים את newChild להורה הנוכחי (השורש).



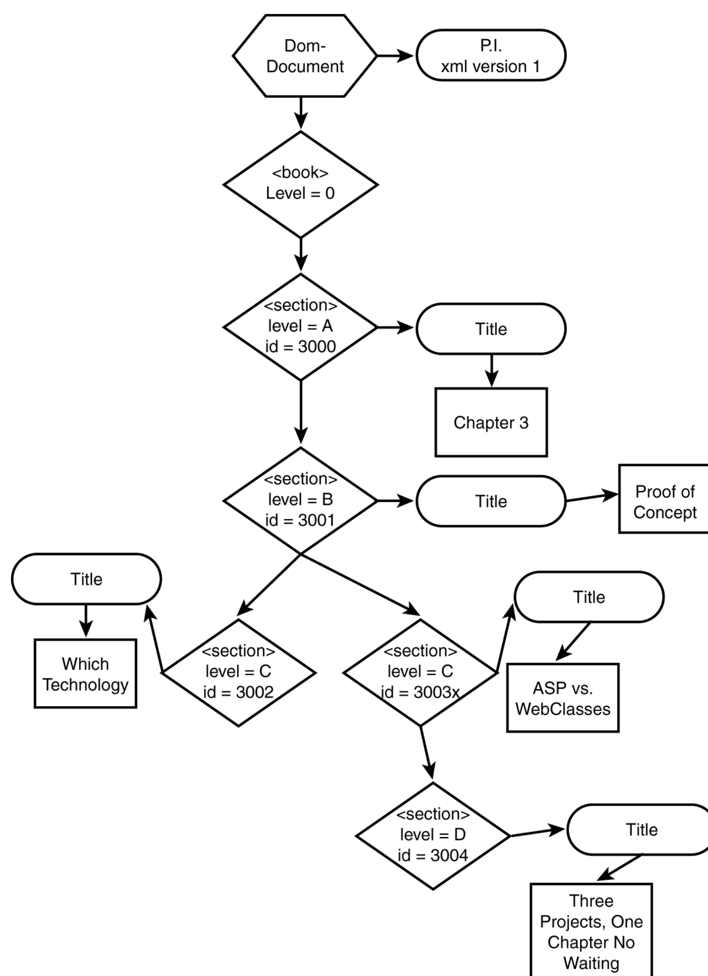
תרשים 5.5: לאחר שרשור newChild

לאחר הוספת הרכיב החדש, אנו מדלגים מטה לשורה 81, וקובעים את המשתנה המקומי curParent להצביע על הרכיב החדש (הרכיב שהוספנו זה עתה). אחר כך אנו קובעים את הדגל omit ל-true כך שרכיב זה לא ייווסף שוב. זה גורם לנו לעבור להצהרה Next בשורה 90, ושוב אנו חוזרים לראשית הלולאה for בשורה 36.

לאחר שנוספו חמישה קטעים, מסמך DOM הפלט נראה כמו העץ המוצג בתרשים 5.6.

כאן אנו מתחילים לראות הירכית הכלה. היחס בין המחלקות מבוטא היטב בתרשים עץ, אך תרשים הכלה המוצג בתרשים 5.7, ממש מבהיר את הנקודה.

קטע D הוא כעת בן של **ומוכל על ידי** קטע C. זהו בדיוק היחס אליו שאפנו להגיע. שימוש בדיאגרמה של תיבות בתוך תיבות הופך ללא יעיל, ולכן נוח יותר לחשוב במונחים של עץ, כך שתרשים 5.4 הוא דרך נפוצה יותר להבעת יחסים אלו. למרות זאת, הבעיה עם עץ היא שקל לאבד את ההכלות המרומזות בהירככיה.

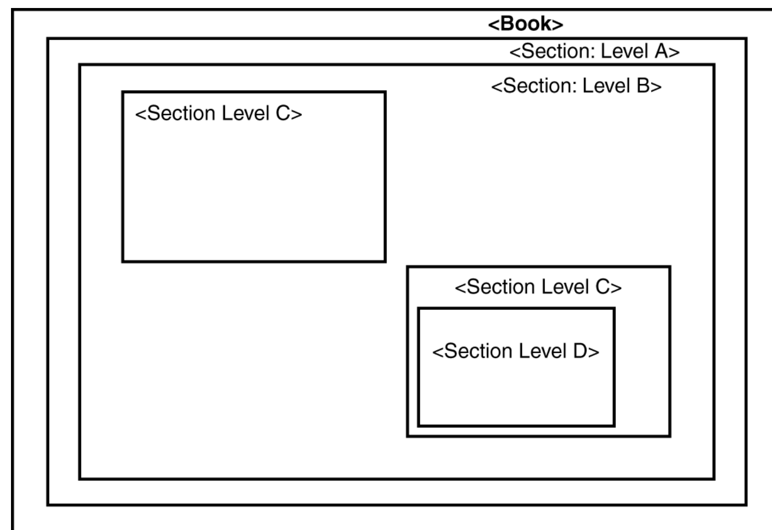


תרשים 5.6: לאחר הוספת חמישה קטעים.

איתור רכיבים

כל רכיב במסמך DOM הפלט שלנו יכול להיות מזוהה בדיוק ובפשטות במונחים של הקשר שלו למסמכים אחרים. בדיוק כשם שאני הבן השני של הבן הרביעי של סבא שלי, כך הרכיב "Three Projects, OneChapter, No Waiting" הוא הבן הראשון של הבן השני של הכותרת ברמה-B "Chapter 3".

כאשר עץ זה מושלם, הוא ישקף ישירות ומפורשות את היחסים בין "articles" (מאמרים) או קטעים שבספר. מדיאגרמה זו אנו יכולים לבנות בקלות טבלה של תוכן העניינים (ונעשה זאת בהמשך!), ונהיה מסוגלים לנווט בין הקטעים בעקביות גם לוגית וגם אינטואיטיבית.



תרשים 5.7: דיאגרמת הכלה

אחים ודודים (Siblings and Aunts)

התחשבנו במקרה של הכנסת רכיב ברמה-C תחת רכיב ברמה-B בשורה 64. כיצד הצלחנו להכניס את רכיב האח ברמה-C המוצג בדיאגרמה? במקרה זה רמת הרכיב החדשה תהיה שווה בדיוק לרמת "curParent's". כלומר, לאחר שהכנסנו את הרכיב הראשון ברמה-C, הוא הפך להורה הנוכחי. כאשר הרכיב השני ברמה-C יופיע, ההצהרה if בשורה 67 תהיה אמת:

```
67:        ElseIf newC.getAttribute("level") = curParent.getAttribute("level") Then
```

במקרה זה, בשורה 69 אנו מאחזרים את צומת ההורה לרכיב של curParent (כלומר, אנו משיגים את הרכיב ברמה-B) ומשרשרים לו את הרכיב החדש (הרכיב ברמה-C השני שלנו), ובכך מוסיפים בן שני לאוסף הבנים שלו:

```
69:        curParent.parentNode.appendChild newC
```

כשאנו מוסיפים את רמה-D לאותה רמת-C השנייה אנו חוזרים לתנאי הראשון המובא בשורה 64. הרכיב ברמה-D החדש הופך ל-curParent. מה קורה עכשיו כשניתקל ברמה-C? זה הזמן לקפוץ מעלה במעלה העץ, מ-D כל הדרך חזרה ל-B כך שנוכל להוסיף את רמה-C. קוד זה מוצג בשורות 72-76:

```
72:                Do
73:                    Set curParent = curParent.parentNode
74:                    'go until we find a peer
75:                    If curParent.getAttribute("level") = newC.getAttribute("level")
                       ➥Then Exit Do
76:                Loop
```

נמשיך להשוות את curParent לרכיב החדש. אם הרכיב החדש מרמה גבוהה יותר אז אנו קובעים את curParent להיות ההורה של curParent. כלומר, אנו מצביעים על D והרכיב החדש הוא, נאמר C, אנו קובעים את curParent להורה של D (הרכיב ברמה-C שהוא ההורה שלו). אנו ממשיכים לעשות זאת עד שאנו נמצאים מספיק גבוה בעץ כדי להיות מסוגלים להוסיף את הצומת החדש.

נזכור כי הצהרת if הראשונה, בשורה 38, מבצעת בדיקה הרואה אם אנו עובדים עם רכיב. אם היא נכשלת, אנו עוברים לשורה 89. Omit נקבע בתחילה להיות false, והוא נקבע ל-true רק בתוך ההצהרות if, כך ששוב, אם הצומת הנוכחי אינו רכיב אנו בשורה 89 ו-omit הוא false. במקרה זה, אנו רוצים ליצור עותק של הצומת הנוכחי ממסמך הקלט ולשרשר אותו לרכיב הנוכחי במסמך הפלט. הערך true הבוליאני יוצר העתקה "עמוקה", במסגרתה מועתקים הרכיב וכל בניו וצאצאיו:

```
89:        If Not omit Then curParent.appendChild c.cloneNode(True)
```

כאשר הלולאה מסתיימת ואנו עוברים לשורה 92, הספקנו כבר להעתיק את כל הרכיבים למסמך DOM הפלט החדש. כעת אנו יכולים להסיר את התכונה "level" מצומת השורש שלנו. סיימנו עם צירוף הקטעים יחד, ואנו יכולים להחזיר את מסמך DOM הפלט לדף ה-ASP שביצע את הקריאה. הדבר מחזיר אותנו לשורה 11 בתדפיס 5.5:

```
11:     Set outDom = CollectSections(inDom)
```

הפקודה הבאה בשיטה Convert היא בשורה 14:

```
14:     CollectContig outDom, "code"
```

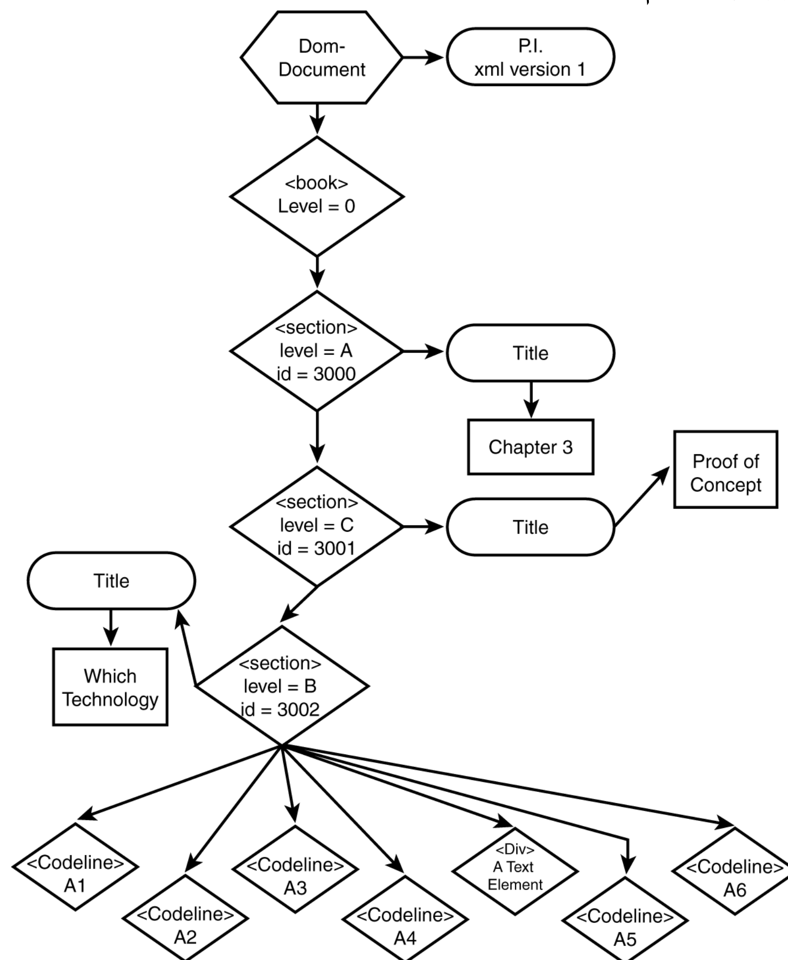
מובילה אותנו למחצית השנייה של השקעת מאמצינו, צירוף הכניסות הרציפות <codeline> לבלוקי <code></code>.

בלוקי קוד

טרנספורמציית ה-XSL סימנה את כל שורות הקוד בתגית <codeline></codeline> כמוצג בתדפיס 5.1. עתה משימתנו היא להבליט את הבלוקים של הקוד ולסמן אותם בתגיות <code></code>. המטרה שלנו היא לאפשר לגזור, להעתיק, להדביק, לשלוח בדואר אלקטרוני, להציג, ובכלל לשלוט באותם בלוקים של הקוד.

לפני שנבחן את הקוד בפירוט, הבה נחזור על הגישה הכללית. נתחיל עם כל שורת קוד כרכיב אינדיבידואלי, כפי שניתן לראות בתרשים 5.8.

הרי לפניכם עובדה מעניינת שנוכל לנצלה אודות רכיבים אלו: כל רכיב יודע מיהו ההורה שלו ומיהו אחיו הקודם. כך, שורת הקוד A3 יודעת שההורה שלה הוא Section ID 3002, ושאחיה הקודם הוא Codeline A2. באופן דומה, Codeline A5 יודעת שאחיה הקודם הוא רכיב טקסט.

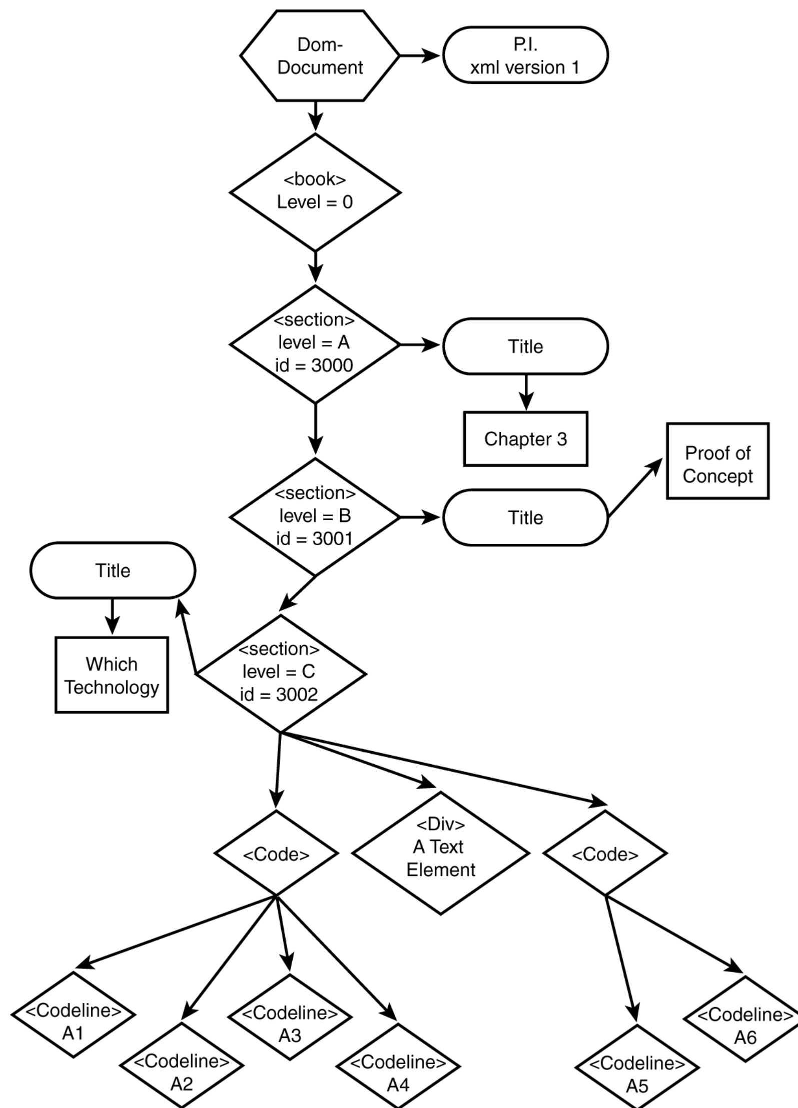


תרשים 5.8: כל שורת קוד היא רכיב

לכן אנו יכולים לציין שבלוק קוד חדש מתחיל בשורת קוד שאחיו הקודם הוא **לא** רכיב שורת קוד. למשל, Codeline A5 מתחיל בלוק חדש, כמו גם Codeline A1.

המטרה שלנו, כשנסיים עם הטרנספורמציה הבאה, היא ליצור מבנה שנראה יותר כמו זה שבתרשים 5.9.

הדרך הברורה והישירה להשיג זאת תהיה להכניס רכיב קוד לתוך המבנה, ואז פשוט להעביר את כל שורות הקוד מתחתיו. לרוע המזל, זה לא יעבוד. אם נעביר את Codeline A1 אל מתחת לרכיב הקוד החדש שלנו,תיווצר בעיה כאשר נבקש מ-Codeline A2 את אחיו הקודם. במקרה כזה, נקבל תשובה שגויה (Codeline A1 לא יהיה שם יותר!).



תרשים 5.9: מבנה המטרה שלנו

לכן מה שאנו חייבים לעשות הוא **להעתיק** את Codeline A1 אל מתחת לרכיב הקוד החדש, ולבקש מ- A2 את אחיו הקודם (שעדיין יהיה A1). כעת משאנו יודעים ש- A2 הוא חלק מאותו בלוק של קוד, אנו יכולים למחוק את A1 המקורי, כך שהוא יופיע רק כילד של רכיב הקוד החדש.

כדי להשיג זאת, נתחיל ביצירת אוסף של כל הצמתים המתויגים כ-codeline.
כעת נוכל להמשיך בעבודה ולחפש באוסף שיצרנו, אחרי רכיבי שורות קוד שאחיהם הקודמים אינם שורות קוד. אחר כך ניצור רכיב קוד ונעתיק את רכיבי שורות הקוד להיות ילדים של אותו רכיב קוד חדש, תוך מחיקתם עם ההתקדמות לאחים הבאים.

תדפיס 5.8 השיטה CollectContig

```
'Combine all the contiguous elements into one parent element
'we do this for a few different kinds of things, so we make the code generic
'the tag name for the individual elements is, e.g. <codeline> while the
 ➥parent element is <code>
'in this case we pass "code" as elName

'here we manipulate the tree in place (vs. above)
Private Sub CollectContig(ByRef inDom As DOMDocument, elName As String)
    Dim baseElems As IXMLDOMNodeList, c As IXMLDOMNode
    Dim prevEl As IXMLDOMElement, lastIndex As Long, curParent
    ➥As IXMLDOMElement

    'get a collection of all xxxline elements
    Set baseElems = inDom.getElementsByTagName(elName & "line")
    For Each c In baseElems
        'is this element the next sibling of the previous one?
        If Not c.previousSibling Is prevEl Then
            'not contigous - create new parent
            Set curParent = inDom.createElement(elName)
            'and insert it as a sibling of the element we are looking at
            'that puts it in the right place - and it is inserted before,
            ➥so we don't interfere
            'with the later siblings
            c.parentNode.insertBefore curParent, c
        End If

        'put a copy of the current element as a child of the current parent
        'it has to be a copy, since we still have to check to see if the next element
        'is a next sibling
        curParent.appendChild c.cloneNode(True)

        'we're now done with the previous element, so delete it
        If Not prevEl Is Nothing Then prevEl.parentNode.removeChild prevEl
```

```
30:
31:        'and keep a pointer to the current element for checking the next one
32:        Set prevEl = c
33:    Next
34:
35:    'delete the last prevEl, if any
36:    If Not prevEl Is Nothing Then prevEl.parentNode.removeChild prevEl
37:
38: End Sub
```

לשיטה CollectContig מועברים שני פרמטרים. הראשון הוא מסמך ה-DOM (Document DOM). חשוב לציין שמסמך ה-DOM המועבר כאן הוא מסמך DOM **הפלט** שנוצר על ידי CollectSections.

בעוד שהשיטה הקודמת, CollectSections, עבדה עם יצירת מסמך DOM פלט חדש, השיטה הנוכחית, collectContig, עובדת באמצעות שינוי הרכיבים של מסמך ה-DOM **בְּמָקוֹם**. אם נחזור לאנלוגיה הקודמת שלנו, כאן אנו מסדרים מחדש את הספרים על גבי המדפים של כוננית הספרים, במקום להעביר אותם לכוננית ספרים חדשה.

מכיון שאנו עומדים לבצע כמעט את אותה העבודה על קטעי ההערות, הפכנו את הקוד הזה לפולימורפי: כלומר, נשתמש באותה הפונקציה לשני המקרים, אך נבחין בין השניים באמצעות הפרמטר. לכן, הפרמטר השני המועבר לשיטה זו הוא שם הרכיב, במקרה זה "code" (עבור המקרה השני, נעביר את "note").

בשורה 11, אנו משרשרים את המחרוזת "line" לפרמטר הזה, ובכך יוצרים "codeline" (או "noteline"). מחרוזת חדשה זו מועברת כפרמטר ל-getElementsByTagName, אשר תחזיר רשימה של צמתים עם השם "codeline".

אוסף זה מאוחסן ב-baseElems. המשתנה המקומי c פועל כאיטרטור על אוסף זה; בכל מחזור של הלולאה for בשורה 12 המשתנה c מקבל את ערך הצומת הבא באוסף baseElems.

בשורה 14 אנו משווים את האח הקודם של הרכיב הנוכחי עם משתנה מקומי prevEl. תנאי ה-if אומר: "השווה את ה-previousSibling של הצומת הנוכחי עם הרכיב שזה עתה סיימנו לעבוד אתו. האם הם זהים? אם לא, זהו בלוק חדש של שורות קוד (CodeLines)".

אם זהו **כן** בלוק חדש, אז אנו יוצרים רכיב <code> בשורה 16 ומשייכים אותו למשתנה curParent.

הבה נניח שבסיבוב הריצה הראשון של לולאה זו, c מתייחס לשורת קוד A1 מתרשים 5.6. prevEl הוא ריק, כך שאנו יודעים ש- A1 פותח בלוק קוד חדש.

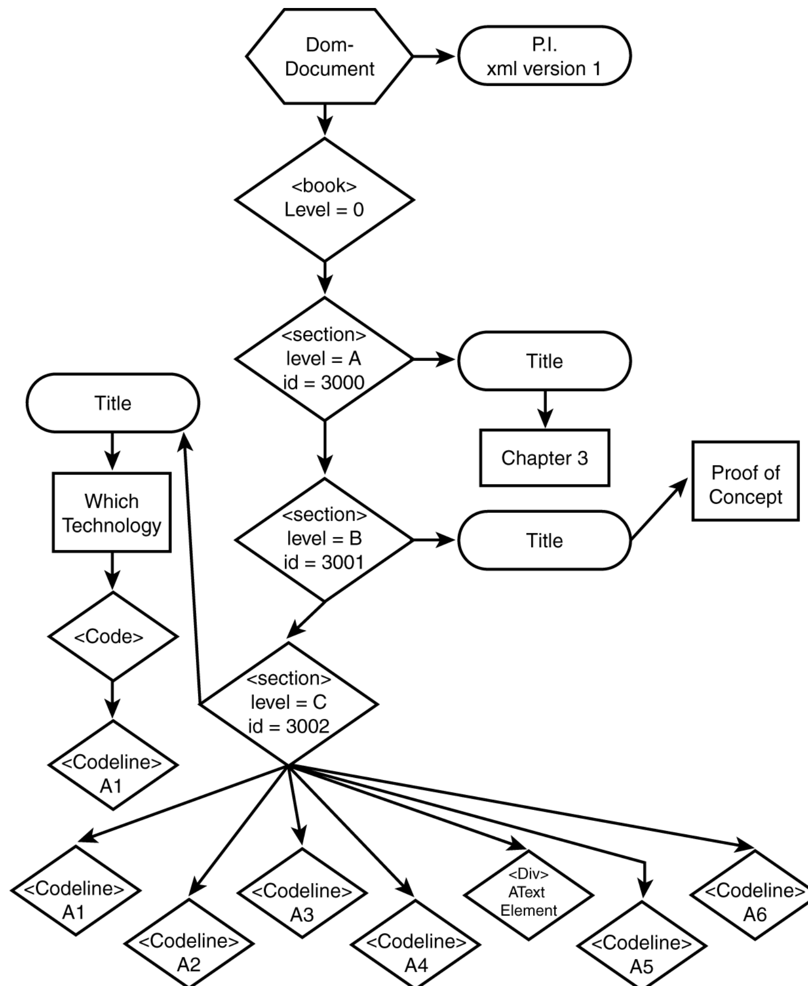
אנו נכניס את רכיב הקוד החדש הזה להיררכיה כאח של רכיבי שורות הקוד. אחר כך נבצע את העבודה הנדרשת להעברת רכיבי שורות הקוד רמה אחת למטה, כך שיהפכו לבנים של רכיב הקוד החדש הזה.

הכנסת הרכיב החדש

אנו רוצים להכניס את רכיב הקוד החדש באותו מיקום בהיררכיה בו נמצאת כרגע שורת הקוד. כדי לעשות זאת, נהפוך את רכיב הקוד החדש לרכיב בן של ההורה של A1. אנו עושים זאת בשורה 20. אנו אומרים להורה של c להכניס רכיב בן חדש curParent (שהוא רכיב הקוד החדש) לפני הבן c.

בנקודה זו הכנסנו את רכיב הקוד החדש (curParent) כאח של c; כעת אנו צריכים להפוך את c (ואת כל שורות הקוד העוקבות) להיות בנים של רכיב הקוד החדש, curParent.

כדי להשיג זאת, אנו **משכפלים** את c ומשרשרים את העותק החדש שלו כרכיב בן של curParent. בכך הפכנו את c לבן ואח של curParent, כפי שניתן לראות בתרשים 5.10.



תרשים 5.10: c הוא כעת רכיב בן **ו**אח של curParent.



הפרמטר true המוצג בשורה 26 אומר ל-cloneNode ליצור עותק עמוק של c, כולל צאצאיו; במקרה זה הטקסט והתצורה של שורת הקוד.

בנקודה זו, ל-prevEl אין כל ערך, כך שמדלגים על הקטע של then בשורה 29. כעת כש-A1 הוא גם רכיב בן וגם אח של curParent, אנו יכולים לקבוע את prevEl ל-c (A1), כמוצג בשורה 32.

כעת חוזרים לראש הלולאה לסיבוב הריצה השני. הפעם c נקבע ל-A2. ההצהרה if בשורה 14 נכשלת. האח הקודם של A2 הוא A1. מכיון ש-prevEl הוא גם כן A1 (קבענו את ערכו בשורה 32 באיטרציה הקודמת) ערכים אלו שווים ולכן מדלגים על ההצהרה if.

כך אנו מדלגים מטה לשורה 26 ויוצרים עותק של A2 ומוסיפים אותו כרכיב בן של הרכיב <code>. כעת לרכיב הקוד יש שני בנים, שניהם גם אחים של רכיב הקוד.

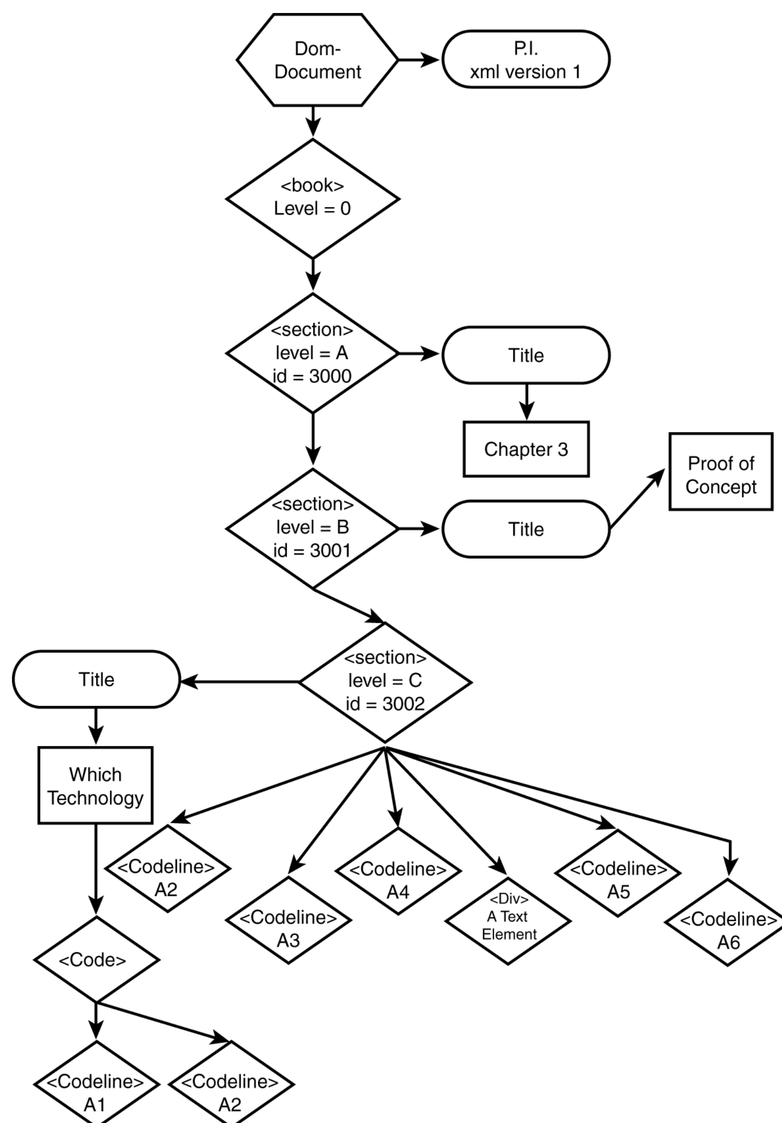
בשורה 29 אנו בוחנים את prevEl, המכיל את שורת הקוד A1. אנו אומרים לצומת ההורה שלו להסיר את הבן prevEl. זה מסיר את A1 מהמיקום המקורי שלו ועכשיו הוא קיים ב-DOM **רק** כבן של רכיב הקוד החדש. עכשיו המבנה נראה כמו בתרשים 5.11.

נוסיף את A3 ואת A4 באותו אופן. בכל פעם שנוסיף בן חדש ל- <code> נבטל את צומת האח הקודם. כך, כשהוספנו את A2 היינו יכולים לבטל את A2 המקורי. כשנוסיף את A3 כרכיב בן של רכיב הקוד, נהיה מוכנים לבטל את A2 הישן.

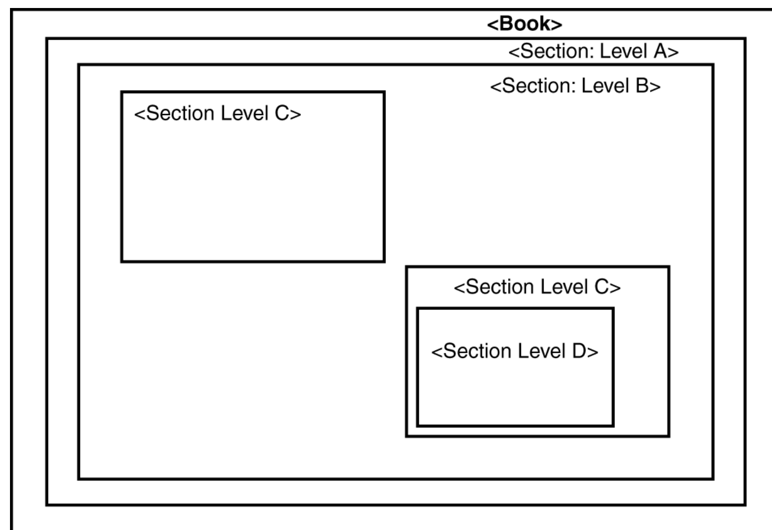
כשנגיע ל-A5 **יהיה** לנו רכיב קוד חדש להוסיף; prevEl יהיה רכיב הטקסט וזה **לא** יהיה האח הקודם של A5 (A4).

את A5 ו-A6 נוסיף כרכיבים בנים של רכיב הקוד השני החדש. העץ יהיה כמעט מושלם, מלבד החיבור של A6 להורה הקודם שלו. אנו מסדרים זאת בשורה 36.

עד שנסיים, כל אחת משורות הקוד כבר תעבור למיקום החדש שלה, תחת רכיב קוד, כפי שניתן לראות בתרשים 5.7 (החוזר לשם הדגשה).



תרשים 5.11: A1 כרכיב בן של הרכיב החדש.



תרשים 5.7: דיאגרמת הכלה.

שורות הערה

זה מחזיר אותנו לשיטה ()Convert, כפי שהוצגה קודם לכן בתדפיס 5.5:

```
13:   'next we collect all the contiguous <codeline> lines into a <code> tag
14:   CollectContig outDom, "code"
15:
16:   'ditto for note lines
17:   CollectContig outDom, "note"
```

לאחר שסיימנו את הקריאה שלנו ל-CollectContig בשורה 14, אנו מוכנים עתה לקרוא לאותה שיטה, CollectContig בשורה 17, הפעם עם המחרוזת "note" במקום "code".

אותה לוגיקה בדיוק תחול גם לגבי כניסות שורות ההערה ב-DOM כפי שעשינו במקרה של כניסות שורות קוד. לא נעבור על זה בפירוט מייגע, שכן אין כל הבדל בין שני המקרים.

כשנסיים, נחזור לשורה 20 בתדפיס 5.5 להשלמת השיטה ()Convert, המוצגת כאן בתדפיס 5.9.

תדפיס 5.9

```
20:   outStr = outDom.xml
21:
22:   'if a DTD has been specified, insert the doctype and external reference
23:   'seems like there should be a way to do it via the DOM, but I can't figure it out
24:   If dtdPath <> "" Then
25:      dtdStr = "<!DOCTYPE book SYSTEM """ & dtdPath & """>" & vbCrLf
```

```
26:         'we want to insert it right after the xml pi
27:         outStr = Replace(outStr, "?>" & vbCrLf, "?>" & vbCrLf & dtdStr, , 1)
28:     End If
29:
30:     'and now we save the results into a file
31:     Open outPath For Output As #1
32:     Print #1, outStr
33:     Close #1
34: End Sub
```

הפקודה הבאה ב-()Convert היא לשייך למשתנה המחרוזת המקומי outStr את תוכן ה-DOM הנכתב כמסמך. **היינו** יכולים לסרוק את העץ, וליצור טקסט לכל רכיב שנפגוש עבור קובץ הפלט שב-XML, אך ישנה דרך קלה יותר.

במקום לסרוק את ה-DOM בעצמנו, אנו יכולים לגשת למאפיין XML של מסמך DOM הפלט. על ידי גישה למאפיין בודד זה, msxml הופך את כל המבנה שלנו חזרה למחרוזת כך שנוכל לאחסן אותו כקובץ.

בזמן שפעולה זו חשפה מאפיין, האובייקט למעשה משטח את ההירכיה כמחרוזת לדיסק.

בכל מקרה, אין לנו רכיב DOCTYPE ל-DTD שלנו. אם חשבת שתוכל להשיג זאת פשוט על ידי שיטה כלשהי כמו-createDocTypeElement אז, לצערנו, טעית. מאיזושהי סיבה זה עדיין לא מופיע בהמלצת ה-XML.

הפתרון שלנו אולי פחות יפה, אך ישים. אנו פונים למאפיין XML כדי לפלוט את כל הקובץ למחרוזת. אחר כך, בשורה 24 אנו משנים ידנית את המחרוזת dtdStr כדי להכניס את הרכיב DOCTYPE מיד לאחר הוראת העיבוד ב-XML שלנו. התוצאה היא שמסמך XML הפלט יתחיל ב:

```
<?xml version="1.0"?>
<!DOCTYPE book SYSTEM "canon.dtd">
```

בנקודה זו המחרוזת שלנו מושלמת אך היא עדיין לא קובץ בדיסק. בשורה 31 אנו פותחים קובץ, בשורה 32 אנו פולטים את מחרוזת מסמך ה-XML, ובשורה 33 אנו סוגרים את הקובץ. בכך מסתיימת הסברוטינה ()Convert, ואנו חוזרים ל-control.asp, ששורותיו האחרונות מוצגות בתדפיס 5.10.

תדפיס 5.10

```
        oH2X.Convert fn0, xmlFN, dtdFN

        'and we no longer need the intermediate file
        fso.DeleteFile fn0

        XHTML2XML = "Converted " & dataPath & ".xhtml" & " to " & xmlFN
End Function
```

עתה חזרנו מ-Convert בשורה 142, ואיננו זקוקים עוד לקובץ הביניים, אותו אנו מוחקים בשורה 145. לבסוף, בשורה 147 אנו מחזירים הודעה על הצלחה למשתמש, שתוצג על הדפדפן.

הצעדים הבאים

משימתנו להפוך את קובץ ה-Word ל-XML בעל מבנה קנוני סוף סוף הושלמה! המסמך שלנו עומד עתה בדרישות ה-DTD שלנו והוא מסמך XML, שאינו תלוי בתגיות HTML, וכן בנוי היטב וכמובן חוקי.

את אותו תהליך ניתן ליישם גם על כל יתר הפרקים הקיימים בספר שלנו.

לאחר שיהיו לנו כל קבצי ה-XML המתאימים לכל פרקי הספר, נוכל לעבד אותם לפי צרכינו.

בפרקים הבאים נראה כיצד לעבד נתונים מתוך מסמכי ה-XML.

פרק 6
אחסון, הבאה לתצוגה, והצגת הסיפורים

בפרק זה:

- מציאת סיפורים מבפנים החוצה
- יישום
- שמירת הסיפורים למסד הנתונים
- יצירת מסד הנתונים
- הצגת הסיפור
- אחזור סיפור ממסד הנתונים
- גיליון הסגנונות מ-XML ל-HTML
- יישום טרנספורמציית ה-XSL
- הצעדים הבאים

ראשית העברנו את מסמך Word שלנו ל-HTML, אחר כך ל-XHTML וכעת, אחרי כמה סיבובים של שינויים, לצורת ה-XML הקנונית שלנו, התואמת ל-DTD שלנו.

עכשיו משיש לנו את הנתונים בצורה שבה רצינו, מה נעשה איתם? בנקודה זו ישנן מספר אפשרויות, ואנו נבחן את חלקן בהמשך הספר.

מטלה אחת היא לחלק את הספר לסיפורים, שבאופן שרירותי הגדרנו אותם להיות כותרת ברמה-D עם התוכן שלה, או כותרת ברמה-C עם התוכן שלה. כלומר, אם כותרת ברמה-C מכילה שלוש כותרות ברמה-D, יש לנו ארבעה סיפורים: תוכן הכותרת ברמה-C עד לכותרת D הראשונה, ותוכן כל אחת משלוש הכותרות ברמה-D.
בעוד שמטלת החלוקה של פרקי הספר לסיפורים עשויה להיראות שרירותית, היא מאוד קרובה למטלות שכיחות יותר, כמו מציאת סיפורים במאמרי מערכת (Newsletters), עיתונים, ספריות מידע, וכן הלאה.

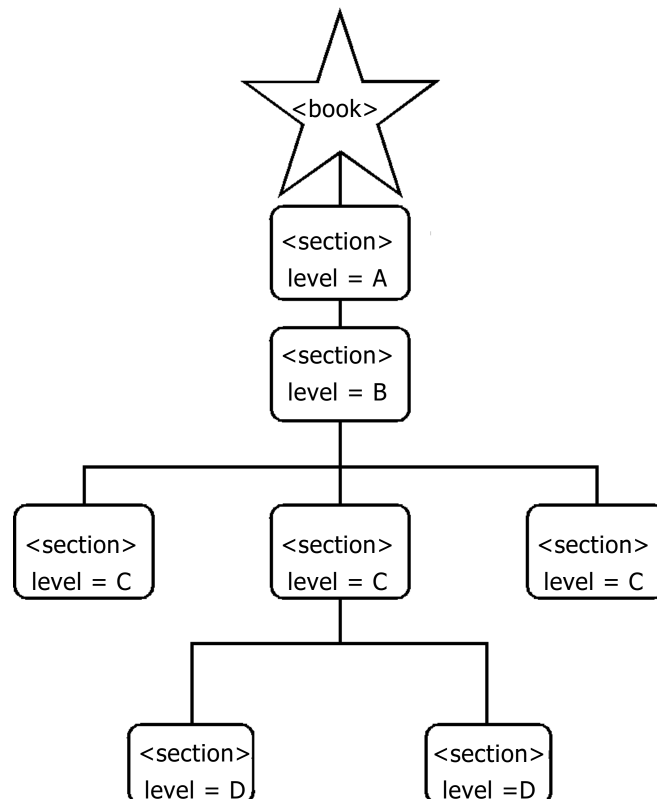
בפרקים הבאים נבחן את הבאת מסמך ה-XML לתצוגה בתצורות שונות, ובכללן דפדפנים מתקדמים (למשל, IE5) כמו גם דפדפנים בסיסיים (למשל, Netscape 3) או הדפסה,פורמט PDF, וכן הלאה. כמו כן, ניצור טבלת תוכן עניינים הניתנת לכיווץ, ובעזרתה נוכל להשתמש למציאת סיפורים במהירות.

לעת עתה, נתרכז בחלוקת הפרקים לסיפורים, היחידות האטומיות של המידע שנאחסן במסד נתונים ואחר כך נציג על פי דרישה.

מציאת סיפורים מבפנים החוצה

נתחיל עם מודל האובייקטים של המסמך, ה-DTD שזה עתה בנינו בפרק 5, ונפרק אותו לרכיבים, תוך אחסון הסיפורים ככל שנתקדם.

המבנה של מסמכים אלו הוא מורכב למדי, להלן סכימה שלו המוצגת בתרשים 6.1.



תרשים 6.1: סכימה של מבנה מסמך

ייצוג מפושט זה של ה-DTD מציג את צומת השורש שלנו, <book> למעלה בראש, עם רכיב בן יחיד מסוג קטע section. לקטע ברמה-A יש רכיב בן ברמה-B, שבתורו יש לו מספר קטעים ברמה-C כבנים. כמובן, שקטע ברמה-A הוא אב לרכיב בן מסוג כותרת

title, אך השארנו זאת מחוץ לתרשים כדי להשאיר את הדברים פשוטים. לקטעים ברמה-C יהיו מספר בנים, ובכללם כותרת, טקסט, ואולי אף הערות וקוד. הקטעים ברמה-C יכולים גם להכיל קטעים ברמה-D, אשר יכולים להכיל הערות, "מונח טכני", קוד וטקסט כרכיבים בנים בפני עצמם.

התחלה עם סיפורים ברמה-D

המפרט מציין שהסיפור ברמה-D כולל את כל הרכיבים המוכלים שלו. הסיפור ברמה-C גם מכיל את כל הרכיבים **שלו**, אך אנו לא רוצים שהסיפורים ברמה-D יופיעו בתוך הסיפורים ברמה-C.

לכן, אם הקטע האמצעי ברמה-C, בתרשים 6.1 הוא "ASP Vs. WebClasses" והוא מכיל טקסט, הערות, קוד ושני קטעים ברמה-D, אנו רוצים שהקטעים ברמה-D יופרדו לסיפורים משלהם.

המפתח ליישום הרעיון הזה הוא להתחיל בתחתית, בסיפורים הפנימיים ביותר: רמה-D. אנו נעביר כל רמה-D (על כל בניה) מה-DOM לתוך מסד הנתונים. אחר כך, כשנגיע לסיפורים ברמה-C, הקטעים ברמה-D המוכלים בהם, לא יהיו שם יותר, וכך נוכל לקחת את מה שנשאר בקטעים של רמה-C ללא כל חשש.

יישום (Implementing Persistence)

הקוד לביצוע האמור לעיל הוא פשוט באופן מפתיע. נתחיל, כמו תמיד, בקובץ ה-ASP שלנו control.asp. תדפיס 6.1 מציג את הפונקציה הרלוונטית.

תדפיס 6.1

```
Function SplitStories()
      dim res, oSplit

      set oSplit = Server.CreateObject("FromScratch.SplitStories")

      res = oSplit.SplitStories(dataPath & ".xml", DSN, DBUser, DBPass)

      SplitStories = "Split " & dataPath & ".xml into " & res & " stories"
End Function
```

שוב אנו מתחילים ביצירת מופע של אובייקט ActiveX; הפעם FromScratch.SplitStories. אנו קוראים לשיטה SplitStories של האובייקט, תוך העברת הקובץ שאנו רוצים לפצל.



כדי שלא תתבלבל, שים לב שפונקציית התסריט (SplitStories), האובייקט (FromScratch.SplitStories), **ושיטת האובייקט** (SplitStories) **הם בעלי אותו** שם. זו לא בעיה; ההבחנה נעשית על פי ההקשר.

נזכור כי בממשק המשתמש אנו מתבקשים לציין את שם הקובץ הבסיסי (למשל, Chap3) שאנו מוסיפים לתיקיה כדי ליצור את הנתיב הבא:

```
dataPath = fso.BuildPath(dataDir, baseName)
```

עכשיו נוסיף את הסיומת xml. כדי ליצור, למשל את Chap3.xml. אנו מעבירים את שם הקובץ ל-SplitStories כפרמטר היחידי שלה.

תדפיס 6.2 מציג את השיטה SplitStories.

תדפיס 6.2

```

'Takes a path to an XML file in our canonical format and splits
 ➥the file up into "stories"
'A story is defined as the contents of an A-D section, NOT including
 ➥any descendant sections
'The resulting stories are stored in the database
Public Function SplitStories(xmlPath As String) As Long
    Dim all As New DOMDocument, i As Long, sections As IXMLDOMNodeList
    Dim c As IXMLDOMElement, level As String, curLev As String,
    ➥curLevIndex As Integer
    Dim numStories As Long, storyId As Long

    numStories = 0

    'open the database so we can store the stories
    DBConn.open "XWDFS", "sa", ""

    'load the XML up into a DOM
    all.async = False
    all.Load (xmlPath)

    'we begin by looking for all the D sections, then C, etc.
    'for each of these, we extract that section's sub-tree from the overall document,
    'create a new XML fragment from it, and store that in the database
    For curLevIndex = 3 To 0 Step -1
        'this will first do D, then C, B, A
        curLev = Chr(Asc("A") + curLevIndex)

        'Create a collection of all sections at this level
        Set sections = all.selectNodes("//section[@level = """ & curLev & """]")

        'we step thru the collection backwards, so that our removals
        ➥don't upset the indexing
        For i = sections.length - 1 To 0 Step -1
            StoreStory sections(i)
```

```
31:            numStories = numStories + 1
32:        Next
33:    Next
34:
35:    DBConn.Close
36:
37:    SplitStories = numStories   'return number of stories
38: End Function
```

בשורה 4 אנו מקבלים פרמטר יחיד: הנתיב לקובץ ה-xml, אח״כ מוגדרים מספר משתנים מקומיים, ובשורה 9 numStories מאותחל לאפס; הוא יספור את הסיפורים שנמצאו כדי שנוכל לדווח על התוצאות.

בשורה 12 אנו מבצעים קישור למסד הנתונים שלנו דרך ODBC.



התיקיות להגדרת ODBC ניתנות בפרק 2, "מעבר מ-HTML ל-XHTML".

יש צורך כמובן במסד נתונים, אותו תוכלו לאחזר מהקבצים שבאתר האינטרנט שלנו, או ליצור בעצמכם. את מבנה מסד הנתונים נסקור מאוחר יותר בפרק זה.

בשורה 15 אנו קובעים את המאפיין async של מסמך ה-DOM ל-false. כפי שצוין בפרקים הקודמים, הדבר יגרום לתוכנית לחכות, עד שהמסמך כולו ייטען לפני שתמשיך. בשורה 16 אנו טוענים את מסמך ה-DOM עם תוכן הקובץ שהועבר כפרמטר (xmlPath). התוצאה תביא ליצירת ה-DOM עבור קובץ ה-XML שלנו.

מרבית העבודה מתבצעת בשורות 21-33.

שורות 21 ו- 23 מאפשרות לנו ״לספור לאחור״ מ-D ל-C, ל-B ומשם ל-A. בשיטת הספירה הזו, ניצלנו את העובדה שהאותיות מסודרות בטבלת ה-ASCII בזו אחר זו. אנו מתחילים עם ערך ה-ASCII של האות ״A״ ומוסיפים לו 3, כדי לקבל D. בפעם הבאה נוסיף רק 2 (כדי לקבל C), אחר כך 1, ולבסוף 0. הנה הדרך: curLevIndex מאותחל ל- 3 ונוסף לערך המספרי השלם של תו ה-ASCII "A"; הערך המתקבל משוערך כתו, ותו זה משויך למשתנה המקומי curLev.

סיור קצר

Visual Basic: **הפונקציה** ()Asc **מקבלת מחרוזת ומחזירה את ערך ה-**ASCII **המייצג את האות הראשונה במחרוזת.** ()Chr **עושה את ההיפך, היא מקבלת קוד** ASCII **ומחזירה מחרוזת המייצגת את הקוד המסופק.**

עכשיו כש-CurLev מחזיק את האות "D" זה הזמן למצוא את כל הקטעים שתכונתם היא "level="D. אנו מצהירים על אוסף שיכיל את הקטעים האלו, הקרוי (כמה מקורי) sections. הוא מוגדר בשורה 5 כ-IXMLDomNodeList, כלומר, אוסף של צמתי XML.

בשורה 26 אנו קובעים את ערכו של sections לערך התוצאה של הקריאה ל-selectNodes על מסמך ה-DOM שלנו. השיטה selectNodes מקבלת תבנית XSL כפרמטר.



שיטה זו היא המקבילה ב-DOM לפקודת התבנית ב-XSL.

אנו בונים את התבנית עם המשתנה curLev שלנו. אם ל-curLev יש את הערך "D" אז הפרמטר selectNodes שלנו יהיה [@level="D"]sections// .



אנו משתמשים ב- // כדי לציין את כל הצאצאים,
במקום להשתמש ב- /, המחפש רק בנים ישירים.

הוא מחפש בכל הצאצאים של השורש אחר רכיבים מטיפוס section בעלי התכונה level שערכה הוא "D", וממקם אותם באוסף sections.

בשורות 29-32 אנו סורקים את האוסף sections, ועבור כל קטע באוסף, אנו קוראים ל-StoreStory.

לולאת for החיצונית חוזרת שוב, הפעם מתווסף 2 לערך של "A" וקביעת CurLevel ל-"C". ברגע שקטעים אלו נאספים לאוסף הקטעים, אנו סורקים את האוסף, ושוב קוראים ל-StoreStory. אחר כך אנו חוזרים על זה עבור רמה-B ורמה-A. שים לב שיהיה רק קטע אחד ברמה-A וקטע אחד ברמה-B שיאוחסן עבור כל פרק.

שמירת הסיפורים למסד הנתונים

בכל ריצה של לולאת for הפנימית המוצגת בשורות 29-31, אנו קוראים ל-StoreStory, ומעבירים איבר קטע בודד. הקוד עבור StoreStory מוצג בתדפיס 6.3.

תדפיס 6.3

```
'Store the subtree identified by the caller in the db
Private Function StoreStory(c As IXMLDOMElement) As Long
    Dim newStoryEl As IXMLDOMElement, parent As IXMLDOMElement
    Dim storyId As Long, parentId As Variant, title As String
    Dim newStory As New DOMDocument

    'add the xml pi
    newStory.appendChild newStory.createProcessingInstruction("xml",
➥"version=""1.0""")

    'we create a new XML element to hold the story
    Set newStoryEl = newStory.createElement("story")
```

```
11:     newStory.appendChild newStoryEl
12:
13:     'we get the storyId from the id of the section
14:     storyId = c.getAttribute("id")
15:
16:     'and the parentId is the id of the parent section, if any
17:     Set parent = c.parentNode
18:     parentId = parent.getAttribute("id")
19:     If IsNull(parentId) Then parentId = 0
20:
21:     'we'll store the title also as a separate field, so it is easy to search for
22:     title = c.selectSingleNode("title").Text
23:
24:     'just in case there are any reserved characters, we'd better quote them
25:     title = HTMLQuote(title)
26:
27:     'now insert the desired subtree to form the body of the story
28:     'note that since we are not copying, we are actually removing c from
        ➥its original parent
29:     'which is what we want, so that later selections at higher levels won't include
        ➥the children
30:     newStoryEl.appendChild c
31:
32:     'and store the XML, along with the extracted metadata, in the data base
33:     'first delete any previous version
34:     DBConn.Execute "delete from stories where StoryId = " & storyId
35:     DBConn.Execute "insert into Stories(StoryId, ParentId, SectionLevel, Title, XML,
        ➥TaglessText) Values(" _
36:         & storyId & ", " & parentId & ", " _
37:         & DBQuote(c.getAttribute("level")) & ", " & DBQuote(title) & ", " _
38:         & DBQuote(newStory.xml) & ", " & DBQuote(newStory.Text) & ")"
39:
40:     StoreStory = storyId    'return the story id of the stored story
41: End Function
```

בשורה 1 ניתן לראות את החתימה של השיטה StoreStory; היא מקבלת פרמטר יחיד, DomElement. מטרתנו היא ליצור מסמך XML חדש ושלם מסיפור זה, ולאחסן מסמך זה במסד הנתונים.

נתחיל בשורה 4 על ידי יצירת מסמך ה-DOM החדש, שנקרא לו newStory. בשורה 7 אנו מוסיפים את הוראת העיבוד הסטנדרטית למסמך ה-DOM החדש שלנו.

בשורה 10 אנו יוצרים רכיב חדש מטיפוס story, הקרוי newStoryEl. שים לב כי אנו יוצרים את הרכיב הזה על ידי קריאה ל-createElement על newStory.

כפי שצוין בפרקים הקודמים, האובייקט היחידי שניצור עם מילת המפתח new הוא DomDocument; כל יתר האובייקטים נוצרים על ידי שיטות ייצור אוטומטיות של המסמך, כמו לדוגמה createElement. למעשה createElement רק מקשרת את הרכיב החדש עם המסמך; היא לא מכניסה אותו לעץ. זהו צעד נוסף, המוצג כאן בשורה 11 עם הקריאה ל-appendChild.

עכשיו יש לנו מסמך DOM, newStory, עם צומת בודד (הוראת העיבוד), ורכיב יתום newStoryEl. בשורה 11 אנו מוסיפים את newStoryEl ל-newStory. עכשיו למסמך ה-DOM, newStory, יש שני צמתים: הוראת העיבוד ורכיב ה"סיפור".

הכנסת הסיפור

אנו מוכנים להכניס את הסיפור הזה למסד הנתונים, אך לפני שנעשה זאת, נשייך מספר ערכים למשתנים מקומיים, כך שנוכל למלא במהירות את השדות השונים של רשומת הסיפור (למשל, title, parentID, וכן הלאה).

כזכור, הרכיב c, היה הרכיב שהועבר מ-SplitStories ל-StoreStories. הבה נניח לעת עתה שזהו רכיב ברמה-D. בשורה 14 אנו מבקשים מהרכיב ברמה-D את התכונה "id" שלו. לכל קטע יש "id" ייחודי לו לאורך כל הספר, כך שניתן לזהות באופן ייחודי כל קטע במסד הנתונים. ה-"id" הנוכחי, נוסף כאשר יצרנו את קובץ ה-XML, כפי שתואר בפרק 4. נחזיק ב-id הנוכחי במשתנה המקומי storyID.

מעקב אחר ה-id

אנו גם רוצים לעקוב אחר ה-id של ההורה בסיפור הזה, כך שנוכל לשחזר את מבנה הספר במידה ונצטרך. אם זהו קטע ברמה-D, ההורה שלו יהיה הקטע ברמה-C שאליו הוא שויך בקובץ ה-XML. אנו קובעים את המשתנה המקומי parent לרכיב ההורה על ידי גישה למאפיין parentNode של c. כל הרכיבים יכולים לדעת מי הם הוריהם, כמו גם מי הם אחיהם הקודמים והבאים.

בשורה 18 אנו מבקשים את ה-id מההורה, ומאחסנים אותו במשתנה המקומי parentID. אם ערך זה הוא NULL, אנו קובעים אותו לאפס בשורה 18. לקטע ברמה העליונה בקובץ זה אין כל הורה, והמוסכמה היא לתת לו ID עם ערך 0. כשאנו יוצרים את טבלת תוכן העניינים, ואנו מוצאים קטעים עם הורה אפס, נדע שהם ההתחלה של הפרק.
בשורה 22 אנו קוראים ל-c.SelectSingleNode.

selectSingleNode הוא בדיוק כמו selectNodes, אלא שהוא מחזיר רק את הצומת הראשון שהוא מוצא, ומחזיר אותו ישירות לרכיב, במקום לאוסף. אנו משתמשים בו כאן מכיוון שידוע לנו שיש רק צומת אחד, וזהו קיצור דרך נוח.

בשורה 22 selectSingleNode מבקשת מ-c להחזיר את הצומת הבודד, התואם לתבנית שהועברה כפרמטר. SelectSingleNode מקבלת כפרמטר תבנית סגנון XSL. במקרה זה אנו מעבירים את השם המדויק של הצומת שאנו מחפשים, "title". SelectSingleNode

תחזיר את הצומת הראשון התואם לתבנית שלנו; במקרה זה ישנו רק צומת אחד שיתאים: הכותרת של הרכיב ברמה-D.

ברגע שיש לנו את רכיב הכותרת, אנו משתמשים במאפיין הטקסט, המחזיר את השרשור של כל צמתי הטקסט הצאצאים תחת רכיב זה.



אם הכותרת היתה במקור "The American Dream", אז לרכיב הכותרת יהיו שלושה בנים: שני צמתי טקסט ורכיב הדגשה. לרכיב ההדגשה יהיה צומת טקסט "American", מאפיין הטקסט של title **יסרוק** את תת העץ הזה, **ישרשר** את רכיבי הטקסט, **ויחזיר** את המחרוזת "The American Dream".

בשורה 25 אנו מעבירים את title לפונקציית עזר שכתבנו, הנקראת HTMLQuote, ומוצגת בתדפיס 6.4. מאפיין הטקסט מפריד יישויות. אם היה לוכסן בטקסט המקורי הוא יאוחסן כ-& כשניצור את מסמך ה-XML. מאפיין הטקסט יחזיר אותו עכשיו כ-&, אך אנו חייבים להחזיר אותו חזרה ל-&.

תדפיס 6.4

```
'expands the reserved characters into their entity representation
Public Function HTMLQuote(ByVal s As String) As String
    s = Replace(s, "&", "&amp;")    'must be first
    s = Replace(s, "<", "&lt;")
    s = Replace(s, ">", "&gt;")
    s = Replace(s, "'", "&apos;")
    s = Replace(s, """", "&quot;")
    HTMLQuote = s
End Function
```

כפי שניתן לראות, HTMLQuote מחליפה תווים שמורים כמו > ו- < בייצוג היישויות שלהם, < ו- > בהתאמה.

בחזרה לתדפיס 6.3, בשורה 30 אנו משרשרים את c, הרכיב שהועבר מ-SplitStories, לרכיב החדש שלנו newStoryEl. זה **מעביר** את c מהמיקום המקורי שלו במסמך ה-DOM המקורי, לרכיב החדש במסמך ה-DOM החדש. כאילו שתלשנו אותו מהעץ הישן והעברנו אותו לעץ החדש.

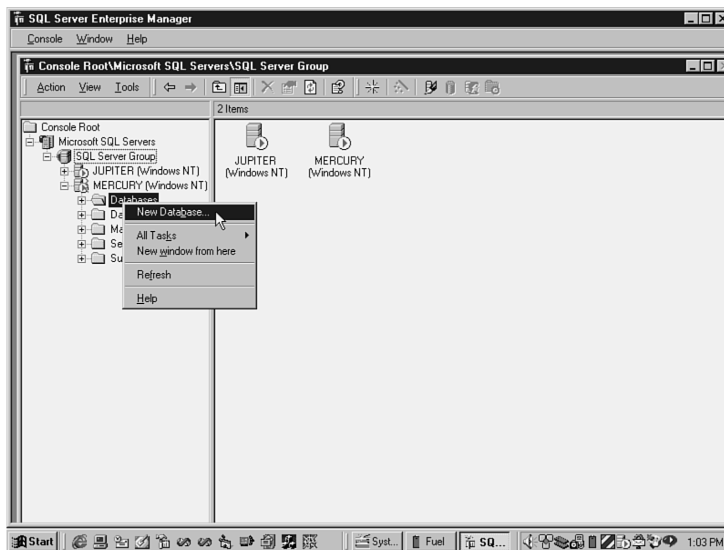
חשוב להבין שכשאנו מעבירים את c הוא מביא אתו את כל הצאצאים שלו. לכן, אם c הוא רכיב ברמה-D ויש לו צמתים בנים, כמו הערות, טקסט, קוד, וכן הלאה, הם כולם יועברו מה-DOM הישן שלהם לחדש.

התוצאה בפועל היא שיש לנו עכשיו מסמך DOM חדש המייצג את הסיפור, ומורכב מהתוכן של הקטע ברמה-D שלנו. באותה מידת חשיבות, הקטע הוסר ממסמך ה-DOM המקורי, וזה מה שרצינו; כשנסיים את עיבוד כל הקטעים ברמה-D ונפנה לעיבוד הקטעים ברמה-C, לא ייוותרו בהם שום קטעים ברמה-D; זה בדיוק מה שהמפרט דורש.

יצירת מסד הנתונים

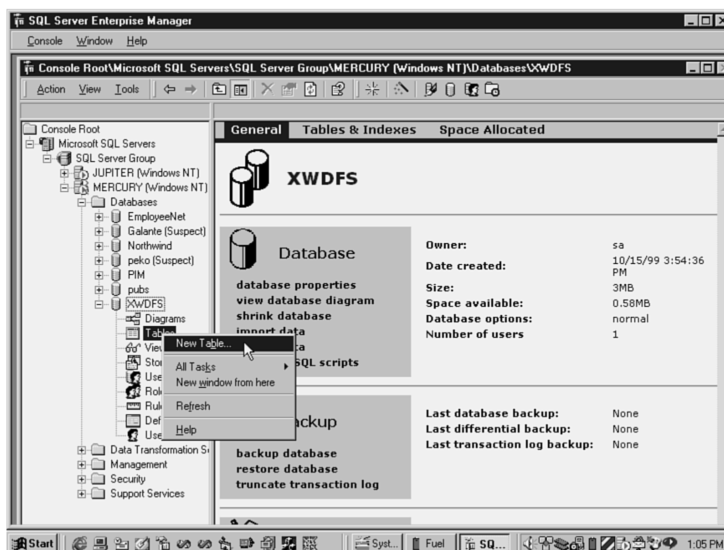
הצעד הבא, כפי שניתן לראות בשורות 34-38, הוא לאחסן את הסיפור החדש במסד הנתונים שלנו. אנו משתמשים במבנה מסד נתונים פשוט מאוד, שאתם יכולים לשכפל על ידי ביצוע הצעדים הבאים:

1. פתח את SQL Server 7 Enterprise Manager.
2. עבור אל רשימת מסדי הנתונים שלך, ולחץ על הכפתור הימני כדי לבחור ב-"new Database", כמוצג בתרשים 6.2.



תרשים 6.2: יצירת מסד נתונים חדש.

3. תן למסד הנתונים את השם "XWDFS" (עבור XML Web Documents from Scratch) ובחר בכל ברירות המחדל של מסד הנתונים.
4. כשמסד הנתונים נוצר, עבור אליו, פרוש אותו כדי למצוא את הטבלאות, ולחץ עם הכפתור הימני על טבלאות כדי לבחור באפשרות "new Table", כפי שניתן לראות בתרשים 6.3.



תרשים 6.3: יצירת טבלה חדשה.

5. תן לטבלה החדשה את השם "Stories".

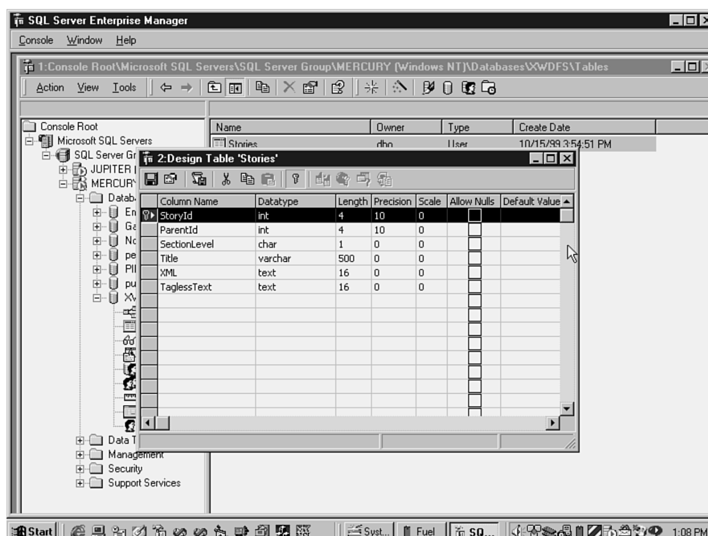
6. הוסף את השדות הבאים, כמוצג בתרשים 6.4.

- StoryID, מספר שלם (integer).
- ParentID, מספר שלם.
- SectionLevel, תו (character) באורך 1.
- Title, מטיפוס varChar באורך 500.
- XML, מטיפוס טקסט.
- TaglessText, מטיפוס טקסט.

בטל את הסימון ליד Allow Null בכל שדה, והפוך את StoryID למפתח הראשי (Primary Key) של הטבלה. שמור את הטבלה החדשה.

זה הכל. טבלה יחידה זו תהיה מספקת לאחסון הסיפורים שלנו. בחזרה לתדפיס 6.3, בשורה 34 אנו מוחקים כל רשומה קיימת עבור הסיפור החדש שלנו. אם אנו מבצעים עדכון, אנו רוצים להיות בטוחים שהסרנו את כל הנתונים הישנים; אם אין לנו נתונים ישנים דבר לא יתבצע. בשורות 35-38 אנו מכניסים את הרשומה החדשה:

```
34:    DBConn.Execute "delete from stories where StoryId = " & storyId
35:    DBConn.Execute "insert into Stories(StoryId, ParentId, SectionLevel, Title, XML,
       ➥TaglessText) Values(" _
36:       & storyId & ", " & parentId & ", " _
37:       & DBQuote(c.getAttribute("level")) & ", " & DBQuote(title) & ", " _
38:       & DBQuote(newStory.xml) & ", " & DBQuote(newStory.Text) & ")"
```



תרשים 6.4: עיצוב "Stories".

אנו מעבירים את storyID, parentID ו-Title שהשגנו קודם לכן, בשורות 14-22. את רמת הקטע של הרכיב לא שמנו במשתנה מקומי, ולכן נשיג ערך זה בשורה 37 על ידי קריאה ל getAttribute על c עצמו.

ערכי המחרוזות מעובדים על ידי פונקציית עזר נוספת שכתבנו, DBQuote, המוצגת בתדפיס 6.5.

תדפיס 6.5

```
'formats a string so it can be passed as a SQL argument
Public Function DBQuote(s As String)
    DBQuote = "'" & Replace(s, "'", "''") & "'"
End Function
```

DBQuote פשוט מחליפה את כל המרכאות הבודדות במרכאות בודדות כפולות, ומחזירה את כל המחרוזת בתוך מרכאות בודדות. למשל, אם היינו רוצים להכניס את המילה don't לתוך מסד הנתונים היינו צריכים להכניס את המילה 'don''t'.

לבסוף, בחזרה לשורה 40 של תדפיס 6.3, אנו מחזירים את storyID לפונקציה הקוראת, SplitStories.

```
40:     StoreStory = storyId    'return the story id of the stored story
```

זה מחזיר אותנו לשורה 30 בתדפיס 6.2, היכן שניתן לראות כי אנו למעשה לא משתמשים ב-ID הזה כלל; החזרנו אותו רק לשם הנוחות בזמן איתור שגיאות.

בשורה 31 של תדפיס 6.2 אנו מגדילים את מונה הסיפורים שלנו. אנו רצים בלולאה בשורות 29-31 עבור כל קטע באוסף. כשאנו מסיימים, אנו חוזרים ללולאה החיצונית, המחזירה אותנו לשורה 21, שם אנו ממשיכים לקבוצת הקטעים הבאה; כלומר, אנו עוברים מרמה D-המרמה C-, ולאחר מכן לרמה B-המ ולרמה A-המ.

לבסוף, כשאנו יוצאים מן הלולאה החיצונית בשורה 33, כלומר, כשאנו מסיימים לפצל את הסיפורים והתחשבנו בכל הרכיבים במסמך ה-XML, אנו סוגרים את החיבור למסד הנתונים שלנו, ומחזירים את מספר הסיפורים שמצאנו:

```
30:            StoreStory sections(i)
31:            numStories = numStories + 1
32:        Next
33:    Next
34:
35:    DBConn.Close
36:
37:    SplitStories = numStories   'return number of stories
```

זה מחזיר את השליטה חזרה לקובץ ה-ASP השולט לשורה 157, המוצגת בתדפיס 6.1 המוצג כאן שוב לנוחיותך.

תדפיס 6.1

```
150: Function SplitStories()
151:        dim res, oSplit
152:
153:        set oSplit = Server.CreateObject("FromScratch.SplitStories")
154:
155:        res = oSplit.SplitStories(dataPath & ".xml", DSN, DBUser, DBPass)
156:
157:        SplitStories = "Split " & dataPath & ".xml into " & res & " stories"
158: End Function
```

נזכור כי היינו בפונקצית התסריט SplitStories שב-control.asp. בשורה 155 מוחזר מספר הסיפורים למשתנה המקומי res, והפונקציה מחזירה את המחרוזת & " Split" "dataPath & ".xml into " & res & " stories, כך שאם עיבדנו את הקובץ Chap3.xml והוא פוצל ל-16 סיפורים, תוחזר המחרוזת הבאה: "Split chap3.xml into 16 stories".

מחרוזת זו מוחזרת לשורה 50 של control.asp, שם היא מוצגת לדפדפן המשתמש על ידי שימוש בשיטת ה-write של אובייקט Response של ASP:

```
50:    Response.Write SplitStories()
```

מה הושג

עתה משראינו את הפרטים, הבה נזנק מספר אלפי מטרים מעלה כדי לקבל תמונה רחבה יותר. Control.asp קרא לפונקציה הפנימית שלו SplitStories, אשר נעזרה בשיטה SplitStories של האובייקט ActiveX שלנו, שהעבירה את הקובץ שאנו רוצים לפצל (Chap3.xml).

השיטה SplitStories הוצגה בתדפיס 6.2. היא פתחה חיבור למסד הנתונים, ואז קבעה לולאה שסרקה את הקטעים ברמה-D, -C, -B ו-A; כאשר הסריקה מבוצעת מלמטה (מרמה-D) כלפי מעלה אל רמה-A. כאשר כל קטע שנמצא, הוזן ל-StoreStories, המוצגת בתדפיס 6.3.

StoreStories חילצה את המידע ההכרחי, ואחסנה את הסיפור במסד הנתונים. SplitStories עקבה אחר מספר הסיפורים שאוחסנו, ודיווחה בחזרה על המספר ל-control.asp, אשר הדפיס את המידע לדפדפן.

כשנסיים, מסמך DOM הקלט שלנו (המבוסס למשל על chap3.xml) יהיה ריק, וכל אחד מהסיפורים הוזן למסד הנתונים.



בעוד שה-DOMDocument רוקן מתוכן, לא נגענו כלל בקובץ המקורי, למשל, Chap3.xml. מה ששונה זה רק הייצוג שלו בזיכרון.

הצגת הסיפור

לאחר שאחסנו את הסיפורים במסד הנתונים, נותרה לנו משימת הצגתם.

מכיון שעכשיו המסמך שלנו מיוצג ב-XML, אנו יכולים להציג אותו במספר תצורות. כמובן, שגישה טבעית ראשונה תהיה להציגו כ-HTML.

המרה מהצורה הקנונית שלנו ל-HTML היא, במספר מובנים, כמו לצעוד לאחור דרך מראת זכוכית. בדרך כלל, נהפוך בחזרה את ההמרות שיצרנו במעבר מ-HTML ל-XML.

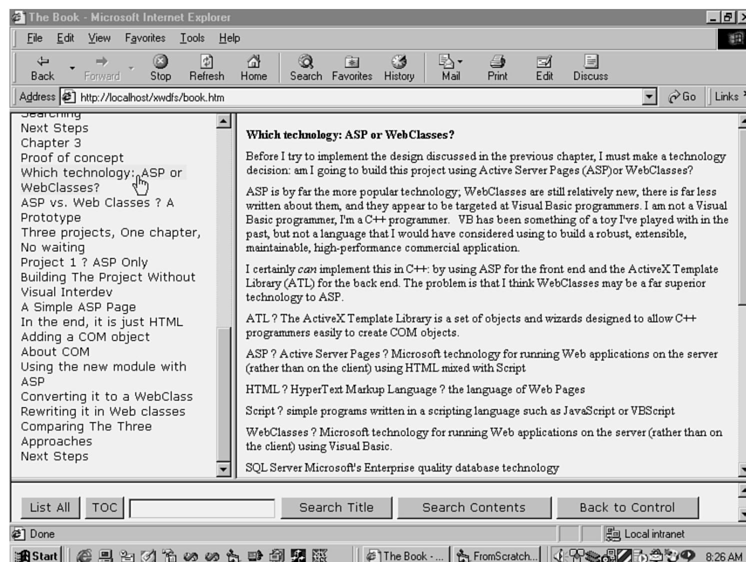
במידה מסוימת, הרבה יותר פשוט לצעוד לאחור, שכן כבר בוצעה עבודה רבה ביצירת מבנה קשיח. המרות רבות הן מיפויים פשוטים מתגיות ה-XML הקנוני שלנו לתגיות <div> ב-HTML.

חשוב להבין שהיינו יכולים כמובן, להמיר כל מספר של סגנונות HTML, התלויים גם בדפדפן שברצוננו לתמוך בו, וגם בהחלטות התצוגה/סגנון הנעשות על ידי מעצב גרפי למשל.

המרה ל-HTML

להזכירכם, מטרתנו הסופית היא ליצור יישום, שבעזרתו נאפשר דפדוף בחלקי הסיפורים שלנו, על גבי הדפדפן. המרת הסיפורים שלנו ל-HTML תסייע לנו במטרתנו זו.

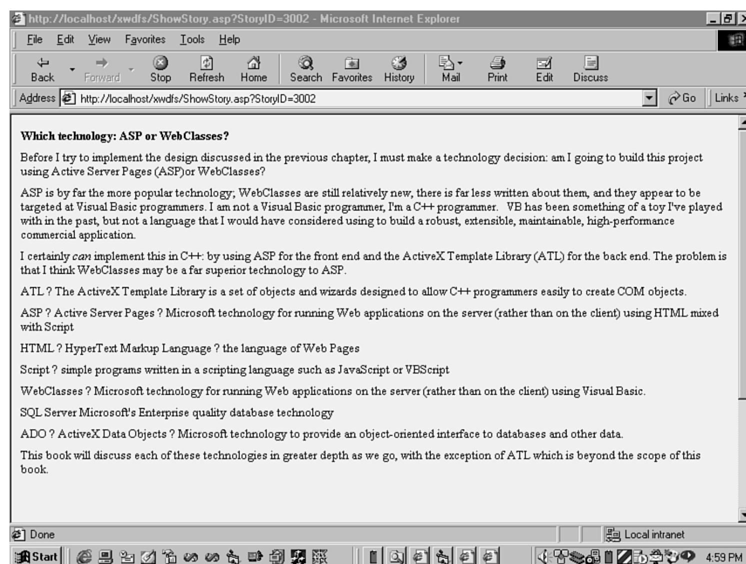
היישום שלנו יאפשר למשתמש בו לסקור את הנושאים במסגרת השמאלית, ולצפות בפרטים של כל סיפור במסגרת הימנית, כמודגם בתרשים 6.5.



תרשים 6.5: צפייה בפרטים של כל סיפור במסגרת

לעת עתה אנו מעוניינים רק במסגרת הימנית. תצוגה זו של הסיפור נוצרת על ידי showstory.asp. אנו יכולים לקרוא לדף ישירות, על ידי העברת ה-id של הסיפור בו אנו מעוניינים. למשל, אם נבחר ב-URL: ...ShowStory.asp?StoryID=3002 נראה את הסיפור 3002 כפי שהוא מאוחסן במסד הנתונים, כמוצג בתרשים 6.6.

עולות כאן שתי שאלות מרכזיות: כיצד השגנו את הסיפור ממסד הנתונים וכיצד הצגנו דף XML כ-HTML? המשך הפרק יתמקד במתן תשובה לשאלות האלו.



תרשים 6.6: סיפור 3002 כפי שהוא מאוחסן במסד הנתונים.

אחזור סיפור ממסד הנתונים

תדפיס 6.6 הוא הקובץ ShowStory.asp במלואו.

תדפיס 6.6

```
0: <!-- #include file ="include.asp" -->
1: <%
2:   'ShowStory.asp - retrieves the XML for single story from the database,
3:   'converts it into HTML via an XSL stylesheet, and displays the results
4:
5:   dim storyId, xml, oXSL, styleSheetName
6:
7:   storyId = Request("storyId")
8:
9:   'get the XML from the database
10:  set rs = DBConn.Execute("select xml from stories where storyId = " & storyId)
11:  xml = rs("xml").value
12:  rs.close
13:
14:  'transform to HTML via the XSL stylesheet
15:  set oXSL = Server.CreateObject("FromScratch.XSLTransform")
16:  oXSL.InputXML = xml
17:  oXSL.XSLFile = fso.BuildPath(codeDir, "Story2HTML.xsl")
18:  oXSL.Transform
19:
20:  'display the HTML
21:  response.write oXSL.OutputXML
22: %>
```

כפי שניתן לראות, אין פה הרבה עניין בקובץ. הוא מתחיל בשורה 0, בהכללת הקובץ include.asp, כפי שהוא מוצג בתדפיס 6.7.

תדפיס 6.7

```
<%
  'General include file for XWDFS

  Option Explicit

  Dim DBConn, rs, fso, DSN, DBUser, DBPass, codeDir, dataDir, baseName, dataPath

  'retrieve the params from the cookie
  codeDir = GetInput("CodeDir")
  dataDir = GetInput("DataDir")
  DSN = GetInput("DSN")
```

```
11:  DBUser = GetInput("DBUser")
12:  DBPass = GetInput("DBPass")
13:  baseName = GetInput("BaseName")
14:
15:  'we almost always need a database connection
16:  set DBConn = Server.CreateObject("ADODB.Connection")
17:  DBConn.Open DSN, DBUser, DBPass
18:
19:  'and also a File System Object to manipulate directories
20:  set fso = Server.CreateObject("Scripting.FileSystemObject")
21:
22:  dataPath = fso.BuildPath(dataDir, baseName)
23:
24:  Function GetInput(name)
25:        GetInput = Request.Cookies("XWDFS")(name)
26:  End Function
27:
28:  %>
```

הקובץ include.asp קיים רק כדי לטפל במספר קבועים הנדרשים על ידי דפי ה-ASP שבשימוש יישום זה. יש לו רק פונקציה אחת, GetInput, המוצגת בשורות 24-26, אשר מחזירה עוגיה (cookie) המבוססת על שם שמסופק לה כפרמטר.

שימו לב כי כל קובץ include מצוי בתוך מצייני התסריט <% %> בשורות 0 ו- 28. אין כאן כלל HTML, רק VBScript בצד השרת. זה נכון גם עבור ShowStories. אחרי השורה הראשונה בה נכלל include.asp, יתר הקובץ, משורה 1 עד שורה 22, הוא תסריט בצד השרת.

בשורה 7 של תדפיס 6.6 אנו אוספים את ה-id של הסיפור המועבר כפרמטר (StoryID=3002 המוצג ב-URL). העבודה מתבצעת בשורות 10 עד 12:

```
10:  set rs = DBConn.Execute("select xml from stories where storyId = " & storyId)
11:  xml = rs("xml").value
12:  rs.close
```

בשורה 10 אנו מבצעים הצהרת SQL הנשלחת אל מסד הנתונים שלנו, ומבקשים את השדה xml מהסיפור התואם ל-storyID המסופק. בשורה 11, אנו משייכים את הטקסט המצוי בשדה xml למשתנה המקומי xml. כמובן, זהו אותו הטקסט המאוחסן במסד הנתונים מ-DOM הפלט שזה עתה סיימנו לעבוד אתו, מוקדם יותר בפרק זה. בקצרה, למשתנה xml יש עכשיו את מחרוזת ה-XML המייצגת את הסיפור, המוכנה להיקרא לתוך מסמך DOM.

יצירת מופע של מסמך DOM הקלט שב-XML

המשימה בשורות הבאות, שורות 15-18, היא ליצור מופע של DOMDocument המתקבל כקלט מה-XML שהשגנו ממסד הנתונים, ולהפוך את ה-XML הזה ל-HTML על ידי שימוש בטרנספורמציית XSL:

```
15: set oXSL = Server.CreateObject("FromScratch.XSLTransform")
16: oXSL.InputXML = xml
17: oXSL.XSLFile = fso.BuildPath(codeDir, "Story2HTML.xsl")
18: oXSL.Transform
```

בשורה 15 אנו יוצרים את אובייקט ActiveX הנקרא XSLTransform, המוכר לנו. בשורה 16 אנו קובעים את המאפיין InputXML שלו למחרוזת xml שזה עתה השגנו ממסד הנתונים. בשורה 17 אנו קובעים את המאפיין XSLFile שלו לגיליון סגנונות XSL המתאים, "Story2HTML.XSL", בו נדון ממש בעוד רגע. לבסוף, בשורה 18 אנו קוראים ל-Transform.

כשנסיים, בשורה 21 נכתוב לדפדפן הלקוח, על ידי העברת המאפיין OutputXML של מסמך DOM הפלט.

שימוש ב-XSL להצגת HTML

הבה נעמיק מעט את הניתוח של העבודה עם FromScratch.XSLTransform. בחזרה לשורה 16 של תדפיס 6.6 נוכל לראות את השורה הבאה:

```
16: oXSL.InputXML = xml
```

מאפיין זה מיושם כפי שניתן לראות בתדפיס 6.8.

תדפיס 6.8

```
0: Property Let InputXML(ByVal newValue As String)
1:     'load the string into a new DOM
2:     Set myInputDOM = ParseIntoDOM(newValue, False, "Input from XML")
3: End Property
```

כל העבודה מבוצעת בשורה 2, היכן שאנו קוראים ל-ParseIntoDom, תוך העברת מחרוזת XML, יחד עם הפרמטרים False ו-"Input from XML". היישום של ParseIntoDom נראה כבר קודם לכן, אך מוצג כאן שוב כתדפיס 6.9.

תדפיס 6.9

```
'worker function to read an input source into a new DOM
'if isURL is true, then data is the text of a URL; else data is an XML string
Private Function ParseIntoDOM(data As String, isURL As Boolean, src As String)
➥As DOMDocument
    Dim dom As New DOMDocument, result As Boolean
    dom.async = False
```

```
5:    If isURL Then
6:        result = dom.Load(data)
7:    Else
8:        result = dom.loadXML(data)
9:    End If
10:   If Not result Then
11:       ReportParseError src, dom
12:   End If
13:   Set ParseIntoDOM = dom
14: End Function
```

ניתן לראות כי isURL הוא false ו-src הוא המחרוזת "Input from XML", בעוד שהפרמטר הראשון, data, הוא ה-XML שהשגנו ממסד הנתונים.

בשורה 3 אנו יוצרים את מסמך ה-DOM החדש, ובשורה 5 אנו בודקים את הקלט כדי לראות האם הוא מגיע מ-URL. אם הוא לא, אנו מתקדמים לשורה 8 כדי לקרוא ל-loadXML על מסמך ה-DOM החדש שלנו, תוך העברת המחרוזת שהשגנו ממסד הנתונים. וכך מסמך DOM הקלט, ה-DOMDocument מתעורר לחיים.

בחזרה לתדפיס 6.6 בשורה 18 אנו קוראים ל-Transform. שוב, ראינו את הקוד הזה בעבר, אך מכיוון שהוא קצר, הוא מוצג פה שוב כתדפיס 6.10.

תדפיס 6.10

```
'the function that does the work - actually transforms input via XSL to output
Public Sub Transform()
   Set myOutputDOM = New DOMDocument
   myInputDOM.transformNodeToObject myXSLDOM, myOutputDOM
   if myOutputDOM.parseError.errorCode <> 0 Then ReportParseError "Output",
   ➥myOutputDOM
End Sub
```

מסמך DOM חדש ריק נוצר עבור הפלט. אחר כך אנו קוראים ל-TransformNodeToObject על inputDomDocument, תוך העברת גיליון ה-XSL ומסמך DOM הפלט, ה-DOMDocument שזה עתה נוצר.

גיליון הסגנונות מ-XML ל-HTML

כל התהליך הוא מכני למדי. אנו משיגים את ה-HTML כמחרוזת טקסט ממסד הנתונים, יוצרים מסמך XML DOMDocument מהקובץ, ואחר כך קוראים ל-transformNodeToObject על מסמך ה-DOM הזה, תוך העברת גיליון ה-XSL שלנו. כפי שניתן לראות, העבודה האמיתית, הקשה, היא בבניית גיליון הסגנונות (Stylesheet) הזה.

יישום טרנספורמציית ה-XSL

את מרבית הטקסט שאנו מקבלים ממסמך DOM הקלט נרצה להציג בדיוק כמו שהוא במסמך הפלט. בדרך אנו חייבים להתמודד עם מספר אתגרים: כיצד אנו צריכים להציג כותרות של קטעים וכותרות? כיצד אנו רוצים להציג הערות? איזה סגנון אנו רוצים להחיל על קוד?

בנוסף, אנו חייבים לנקוט במספר טרנספורמציות כדי להציג את רכיבי ה-XML שלנו ב-HTML סטנדרטי. למשל, אנו חייבים להמיר רכיבי מרווחים וטאבים למרווחים קשיחים, בלתי שבירים.

תדפיס 6.11 מציג את גיליון ה-XSL StoryToHTML השלם, אשר נסקור בפירוט.

תדפיס 6.11

```
<?xml version="1.0"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/TR/WD-xsl"
  xmlns="http://www/w3/org/TR/REC-html40"
  result-ns="">

  <!-- default rule for text nodes - just output the text -->
  <xsl:template match="text()">
        <xsl:value-of/>
  </xsl:template>

  <!-- the root node - output the structure of the HTML page
  ➥and begin the recursion -->
  <xsl:template match="/">
  <html>
  <head>
  <style>
        * {font-family:Georgia; font-size:10pt; margin-bottom:6pt;}
        .unknown {color:red;}
        .code {background-color:lightgrey;padding:3pt;}
        .textcode {font-family: Courier New; font-size:10pt; }
        .codeline, .codeline b{font-family: Courier New; font-size:9pt;
        ➥line-height:9pt; margin-bottom:0; }
        .sectionHeader {font-weight:bold}
        .LH, .FN {font-style:italic;}        <!-- listing header, figure caption -->
        .note {background-color:lightgreen; margin:0 4em; padding:3pt;}
        .noteline, .noteline b, .noteline i, .noteHeader {font-family:Verdana;
        ➥font-size:8pt;}
        .noteline {margin-left:2em;}
        .noteHeader {font-weight: bold;}
        .summary {font-style:italic; font-size:8pt; margin-bottom:0;}
```

```
</style>
</head>
        <xsl:apply-templates/>
</html>
</xsl:template>

<!-- sentinel for any left over tags -->
<xsl:template match="*">
        <div class="unknown">
                !!<xsl:node-name/>!!
                <xsl:apply-templates />
        </div>
</xsl:template>

<!--
        The top level element - might be story or book depending on who's calling
        Just wraps the content inside <body> tags
        Then finishes with some summary info about numbers of nodes
        ➥of different kinds
 -->
<xsl:template match="story|book">
<body>
        <xsl:apply-templates/>
        <div class="summary">
                Section summary
                <br/><xsl:eval>this.selectNodes("//div").length</xsl:eval> paragraphs
                <br/><xsl:eval>this.selectNodes("//code").length</xsl:eval> code blocks
                <br/><xsl:eval>this.selectNodes("//note").length</xsl:eval> notes
        </div>
</body>
</xsl:template>

<!--
        Sections just map into DIVs with their own class. We'll keep
        ➥the level attribute incase we ever
        want to do formatting based on it
 -->
<xsl:template match="section">
<div class="section"><xsl:attribute name="level"><xsl:value-of
➥select="@level"/></xsl:attribute>
        <xsl:apply-templates/></div>
</xsl:template>

<!--
```

```
        Our basic text paragraph element. We maintain the class attribute,
        ➥which is really the same
        as the original Word style name
 -->
<xsl:template match="div">
        <div>
                <xsl:attribute name="class"><xsl:value-of
                ➥select="@class"/></xsl:attribute>
                <xsl:apply-templates/>
        </div>
</xsl:template>

<!-- we ignore PD - these are publishing directions, e.g. "begin note" -->
<xsl:template match="div[@class='PD']">
</xsl:template>

<!--
        These are our specialized content types for code blocks and notes.
        We just change them into divs with corresponding styles
-->
<xsl:template match="codeline|code|noteline">
<div><xsl:attribute name="class"><xsl:node-name/>
➥</xsl:attribute><xsl:apply-templates/></div>
</xsl:template>

<!-- The note element is a little special, because we want to put in a header
➥that says "Note" -->
<xsl:template match="note">
        <div class="note">
        <div class="noteHeader">Note</div>
        <xsl:apply-templates/>
        </div>
</xsl:template>

<!-- Titles for sections also map into their own div class -->
<xsl:template match="title">
        <div class="sectionHeader">
                <xsl:apply-templates/>
        </div>
</xsl:template>

<!-- underline elements are used to denote code in the text - turn these into
➥their own special spans -->
<xsl:template match="u">
```

```
        <span class="textcode"><xsl:apply-templates/></span>
</xsl:template>

<!-- handle vanilla HTML-like tags - just map them to the same thing -->
<xsl:template match="b|i|sub|sup|br">
        <xsl:element>
                <xsl:apply-templates/>
        </xsl:element>
</xsl:template>

<!-- spaceruns and tabs get mapped into sequences of  s. We approximate
➥tabs with 4 spaces -->
<xsl:template match="spacerun">
        <span><xsl:eval>Repeat(" ",
        ➥this.getAttribute("len"))</xsl:eval></span>
</xsl:template>

<xsl:template match="tab">
        <span><xsl:eval>Repeat(" ", 4)</xsl:eval></span>
</xsl:template>

<!-- translate special chars into entities for display in HTML - we use numbered
➥entities so we don't have to
        explicitly define them - may only work in IE -->
<xsl:template match="char">
        <xsl:choose>
                <xsl:when test="@type[. = 'emDash']">&#8212;</xsl:when>
                <xsl:when test="@type[. = 'bullet']">&#8226;</xsl:when>
                <xsl:when test="@type[. = 'ellipsis']">&#8230;</xsl:when>
                <!-- we'll just use 2 nbsps for emSpace - #8195 doesn't seem
                ➥to work as documented -->
                <xsl:when test="@type[. = 'emSpace']">  </xsl:when>
                <xsl:when test="@type[. = 'smartApos']">&#146;</xsl:when>
                <xsl:when test="@type[. = 'smartLQuote']">&#147;</xsl:when>
                <xsl:when test="@type[. = 'smartRQuote']">&#148;</xsl:when>
                <xsl:otherwise><span class="unknown">char: <xsl:value-of
                ➥select="@type"/></span></xsl:otherwise>
        </xsl:choose>
</xsl:template>


<xsl:script>
<![CDATA[
// returns a string consisting of n copies of t
```

```
147:  function Repeat(t, n)
148:  {
149:        var r = "";
150:        while (n-- > 0)
151:              r += t;
152:        return r;
153:  }
154:  ]]>
155:  </xsl:script>
156:
157:  </xsl:stylesheet>
```

מתחילים

גיליון ה-XSL שלנו מתחיל בשורה 0 על ידי זיהוי נאות של עצמו עם הוראת עיבוד. אחר כך רכיב גיליון הסגנונות מזהה את ה-namespaces בהם נשתמש. שורה 2 מזהה את ה-namespace הקרוי xsl כפי שמתואר בפרקים הקודמים. שורה 3 מצהירה על namespace הקרוי html, ושוב נשים לב כי הוא ללא שם (xmlns אינו מלווה בנקודתיים ומזהה). זה מציין כי אם לא מצוין אף namespace אז אנו מתכוונים ל-HTML. לבסוף, הפלט namespace נקבע ל-empty, המציין שהוא ישתמש ב-namespace ללא השם, במקרה זה ב-HTML. מסמך DOM הפלט שלנו יהיה מסמך HTML, כך שזה אכן נכון.

הכותרת העליונה ב-HTML

בשורה 12 אנו מבצעים התאמה לרכיב השורש:

```
12:   <xsl:template match="/">
```

כשאנו מבצעים זאת, אנו מוסיפים מספר רכיבים למסמך DOM הפלט. אלו הן הכותרות שאנו רוצים בקובץ ה-HTML שלנו, והן כוללות את <html> ו-<head> הסטנדרטיות כמו גם רכיב <style>. הרכיב <style> יספק עיצוב CSS (Cascading Style Sheet) למסמך ה-HTML.

למסמך ה-XML המתקבל יהיה רכיב שורש <book> או <story>. ה-XSL המטפל בכך הוא זהה, כפי שניתן לראות בשורות 47-57:

```
47:  <xsl:template match="story|book">
48:  <body>
49:        <xsl:apply-templates/>
50:        <div class="summary">
51:              Section summary
52:              <br/><xsl:eval>this.selectNodes("//div").length</xsl:eval> paragraphs
53:              <br/><xsl:eval>this.selectNodes("//code").length</xsl:eval> code blocks
54:              <br/><xsl:eval>this.selectNodes("//note").length</xsl:eval> notes
55:        </div>
56:  </body>
57:  </xsl:template>
```

אנו מחליפים את הרכיב book או story ברכיב body של HTML, ואחר כך קוראים ל-apply-templates כדי לאסוף את הרכיבים בתוך book או story. אנו נשוב לשורות 50-54 בקרוב.

מכיון ש-XSL היא הצהרתית באופייה, ולא פונקציונלית, יהיה זה נכון יותר לא לעבור על קובץ ה-XSL שורה אחר שורה, אלא לשקול את המשימות השונות שאנו שואפים להשיג ולהתמקד בדרך השגתן באמצעות XSL.

טיפול בקטעים

נזכיר כי מסמך ה-XML מחולק לקטעים, sections, המייצגים את הכותרות ברמה-A, -B, -C, ו-D. לכל קטע משויכת כותרת, ומספר כלשהו של רכיבי טקסט, הערות וקוד.

בשורות 63-66 אנו מעבדים את sections כמוצג בהצהרת ההתאמה של רכיב התבנית :

```
63:   <xsl:template match="section">
```

כשמבוצעת התאמה לקטעים, אנו יוצרים רכיב <div> חדש עבור מסמך DOM הפלט של ה-HTML. אנו מקודדים את התכונה הראשונה, class, ל-"section". אנו רוצים גם לשמור את התכונה level, ולכן אנו יוצרים תכונה חדשה (על ידי שימוש ב-xsl:attribute) וקובעים את ערכה על ידי שימוש בהצהרה value-of כפי שראינו קודם לכן :

```
64:   <div class="section"><xsl:attribute name="level"><xsl:value-of
      ➥select="@level"/></xsl:attribute>
```



יש לזכור כי הסימן @ בתבנית ה-XSL מציין התאמה לערך של תכונה.

בשורה 65 אנו קוראים ל-apply-templates כדי להבטיח שאנו עוברים ברקורסיה על הקטעים, כך שנעבד כל אחד מהרכיבים המוכלים בהם.

למעשה, כל קטע יתחיל ב-div המזהה את הקטע, ובין התגית <div> הפותחת והתגית הסוגרת שלה </div>, תהיה הכותרת, כל הטקסט, ההערות והקוד של הקטע.

הכותרת היא הרכיב הראשון שניתן לראות בקטע. שורות 99-104 מבצעות התאמה לרכיבי כותרת ויוצרות div עם התכונה sectionHeader :

```
 99: <!-- Titles for sections also map into their own div class -->
100: <xsl:template match="title">
101:       <div class="sectionHeader">
102:             <xsl:apply-templates/>
103:       </div>
104: </xsl:template>
```

התכונה "class="sectionHeader תשתמש באמצעות הדפדפן בביצוע התאמה לבורר ה-CSS sectionHeader כמוצג בשורה 21 :

```
21:         .sectionHeader {font-weight:bold}
```

התוצאה היא ש-sectionHeader ישתמש בסגנון ברירת המחדל, המוצג בשורה 16 :

```
16:        * {font-family:Georgia; font-size:10pt; margin-bottom:6pt;}
```

אך ישנה את סגנון ברירת המחדל לקביעת font-weight למודגש.

בשורה 102 אנו קוראים ל-apply-template, אשר תבודד את הטקסט מהכותרת. בשורות 6-9 אנו מציינים כי כל רכיב מטיפוס טקסט ימופה **כפי שהוא** למסמך DOM הפלט ; כלומר הטקסט פשוט יועבר הלאה :

```
7:  <xsl:template match="text()">
8:        <xsl:value-of/>
9:  </xsl:template>
```

זהו הטקסט אשר יוצג בהדגשה. את הקוד הזה בדיוק ניתחנו לפרטיו בדיון הקודם על XSL ולכן לא ננתח אותו גם כאן.

רוב הטקסט שבמסמך הקלט נמצא ברכיבי <div>, שכל אחד מהם מייצג פיסקה של טקסט. בשורות 72-77 אנו מטפלים ברכיבים האלו, פשוט על ידי יצירה של רכיב div חדש עבור פלט ה-HTML והעתקת התכונה class בשלמותה :

```
72: <xsl:template match="div">
73:        <div>
74:               <xsl:attribute name="class"><xsl:value-of
               ➥select="@class"/></xsl:attribute>
75:               <xsl:apply-templates/>
76:        </div>
77: </xsl:template>
```

הבאת הקוד לתצוגה (Rendering Code)

על פי המפרט שלנו, כשאנו מציגים קוד הוא אמור להיות מופרד מגוף הטקסט המרכזי. כדי להשיג זאת, ניצור תגית <div> עבור הרכיבים <code> ו- <codeline> שלנו. לכל אחד תהיה תכונה class משלו, ולכן כל אחד יתאים לבורר CSS ייחודי.

בשורה 87 אנו מבצעים התאמה לכל אחד מה-codeline, code או noteline :

```
87: <xsl:template match="codeline|code|noteline">
```

הפעולה שלנו על כל אחד מרכיבים אלו היא ליצור רכיב div, עם התכונה class. הערך המשויך ל-class הוא ה-node-name שכנגדו בוצעה ההתאמה (למשל, codeline, code או noteline). אחר כך אנו קוראים ל-apply-templates כדי להבטיח שאנו עוברים ברקורסיה על רכיבים אלו :

```
87: <xsl:template match="codeline|code|noteline">
88: <div><xsl:attribute name="class"><xsl:node-name/>
    ➥</xsl:attribute><xsl:apply-templates/></div>
89: </xsl:template>
```

כך, רכיבי קוד יומרו לרכיבי <div> עם התכונה class שערכה הוא code. בשורה 18 ניתן לראות כי הסגנון של רכיבי הקוד הוא לשנות את צבע הרקע וליצור רווח בן 3 נקודות (3pt Padding):

```
18:     .code {background-color:lightgrey;padding:3pt;}
```

לכן, כשנציג את הקוד ב-HTML הוא יוצג על גבי רקע בצבע בהיר, ויבלוט מגוף הטקסט.

טיפול בהערות

האסטרטגיה שלנו לטיפול בהערות היא דומה מאוד, וכבר ראינו ששורה 87 מבצעת התאמה לרכיבי noteline.

בכל אופן, התאמה לרכיבי הערה חייבת להתבצע בנפרד, שכן אנו רוצים לכתוב את המילה Note בראש ההערה, לפני הצגת התוכן של ההערה עצמה. אנו רואים יישום של זה בשורות 92-97:

```
92: <xsl:template match="note">
93:     <div class="note">
94:     <div class="noteHeader">Note</div>
95:     <xsl:apply-templates/>
96:     </div>
97: </xsl:template>
```

כשאנו מבצעים התאמה להערה, אנו יוצרים div חיצוני שהמחלקה שלו נקבעת ל-note. בשורה 23, אנו מגדירים את סגנון ה-CSS עבור מחלקה זו (צבע רקע ירוק בהיר, וריווחים מתאימים):

```
23:     .note {background-color:lightgreen; margin:0 4em; padding:3pt;}
```

אנו מיד מגדירים div פנימי עם המחלקה noteHeader שתוכנה הוא המילה note. NoteHeader חולק את הסגנון שלו עם noteline ועם שורות הערה מודגשות ומוטות, כפי שניתן לראות בשורה 24:

```
24:     .noteline, .noteline b, .noteline i, .noteHeader {font-family:Verdana;
        ➥font-size:8pt;}
25:     .noteline {margin-left:2em;}
26:     .noteHeader {font-weight: bold;}
```

אנו רואים גם סגנונות CSS עבור שורות הערה (המקבלות שוליים שמאליים) וכותרת הערה (המוצגת בהדגשה).

קוד inline

כשמסמך Word המקורי נכתב, מלים מסוימות בתוך הפסקאות סומנו בקו תחתון, במידה והן התייחסו לקוד. למשל, בפרק 3 בספר הדוגמה, כתוב:

Remember to substitute your own computer's name as the server in (currently it is set to myServer)

הקו התחתי מציין לעורך שזהו קוד, שאמור להיקבע בגופן מיוחד. כשזה נשמר ב-HTML, הקו התחתון נשמר, כפי שניתן לראות במובאה הבאה מ-Chap3.htm:

```
<p class=FT>Remember to substitute your own computer's name
➥as the server in
line 15 (currently it is set to <u>myServer</u>).</p>
```

בכל ההמרות שלנו – ל-XHTML, לקובץ ה-XML הזמני, ולצורת ה-canoncial XML שלנו – רכיב הקו התחתי נשמר ולכן הוא נשמר גם במסד הנתונים.

כעת, כשאנו הופכים זאת **חזרה** ל-HTML, נחליט להכיר במשמעות האמיתית של הקו התחתי הזה, כמובאה של קוד. אנו נשנה את הרכיבים "u" שלנו (כלומר, רכיבי קו תחתי) לתגיות span עם המחלקה "textcode":

```
<xsl:template match="u">
    <span class="textcode"><xsl:apply-templates/></span>
</xsl:template>
```

על ידי קביעת המחלקה ל-"textcode" אנו מורים לדפדפן להחיל את הסגנון המיועד בבורר ה-textcode, כפי שניתן לראות בשורה 19:

```
19:        .textcode {font-family: Courier New; font-size:10pt; }
```

את התוצאה ניתן לראות בתרשים 6.7, בו מוצגים הן הדפדפן והן המקור ב-HTML.

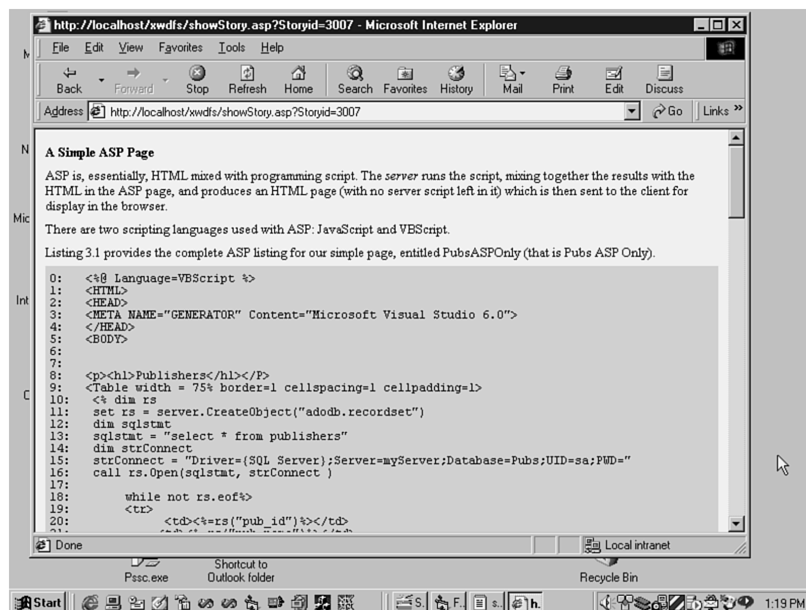


ניתן לטעון כי עשינו בדיוק את ההיפך. כשביצענו המרה ל-XML היינו אמורים לשמור את הכניסות האלו לא כרכיבי u, אלא כרכיבי textCode. זה היה נותן לנו את היכולת לבחור האם להשתמש בגיליונות סגנונות במהלך הדרך.

בכל זאת השארנו זאת בדרך שהוצגה, כדי להדגים מספר נקודות:

1. ההבדל בין התחביר והסמנטיקה הוא לא כל כך ברור.
2. ישנה יותר מדרך אחת להשגת אותה המטרה.
3. לעתים, לא מצליחים מיד בפעם הראשונה.

הנקודה השלישית היא לא בדיחה; כשעובדים עם XML ו-XSL, כמו עם כל שפה מורכבת אחרת, יש מקום לשיקול דעת. לעתים קרובות יש להחליט האם זה משתלם להפוך תגית נתונה לתגית עשירה יותר מבחינה סמנטית; והתשובה לא תמיד כל כך ברורה.



תרשים 6.7: צפייה במקור

HTML בקלות

מסמך ה-XML שלנו מורכב ממספר רכיבים אחרים שחייבים להפוך ל-HTML.

הפשוטות ביותר הן תגיות ה-XML b,i,sub,sup,br שיש להן התאמות ישירות ב-HTML, וניתן להעביר אותן כפי שהן, כפי שניתן לראות בשורות 112-116:

```
111:  <!-- handle vanilla HTML-like tags - just map them to the same thing -->
112:  <xsl:template match="b|i|sub|sup|br">
113:         <xsl:element>
114:                <xsl:apply-templates/>
115:         </xsl:element>
116:  </xsl:template>
```

עבור כל אחד מהרכיבים האלו אנו מגדירים רכיב חדש במסמך DOM הפלט, ומעתיקים את התוכן. להלן אופן הביצוע: הרכיב xsl:element מחזיר את הרכיב הנוכחי. כך, שאם ישנה התאמה לרכיב <b>, אז xsl:element מכניס רכיב <b> במסמך DOM הפלט. בין תגיות הפתיחה והסגירה של xsl:element נראה את התגית xsl:apply-templates, המעתיקה את התוכן של הרכיב <b>.

רווחים וטאבים

למסמך Word המקורי היו רווחים רבים בחלקים שונים במסמך. כש-Word שמר אותם לתוך HTML, הוא שמר אותם על ידי שימוש ברווח בודד המלווה ברצף של תווי a0 הקסה-דצימליים (בבסיס 16). המרנו אותם לרכיבי spacerun שלנו, שהתכונה len שלהם מספרת לנו כמה רווחים נדרשים.

רווחים

כעת, אנו רוצים להחליף אותם ברצף של מרווחים קשיחים (ש-HTML לא יצמצמם לרווח בודד). סימון ה-HTML עבור זה הוא . בשורות 118-121 אנו ממירים את הרווח לרכיבי   :

```
118: <!-- spaceruns and tabs get mapped into sequences of  s. We approximate
     ➥tabs with 4 spaces -->
119: <xsl:template match="spacerun">
120:        <span><xsl:eval>Repeat(" ",
            ➥this.getAttribute("len"))</xsl:eval></span>
121: </xsl:template>
```

בשורה 119 אנו מבצעים התאמה לכל רכיב spacerun. בשורה 120 אנו משתמשים ברכיב xsl:eval כדי להעריך את התוכן של הרכיב eval כתסריט. כך, מנוע ה-XSL יתייחס לכל מה שבין <xsl:eval> ו- </xsl:eval> כאל תסריט, והתוצאה תמוקם כתוכן של תגית ה-span.

התסריט קורא לפונקציה Repeat, תוך העברת היישות המספרית המקבילה ל- כפרמטר הראשון שלה, והערך של התכונה "len" של הרכיב הנוכחי (הרווח).

ניתן לראות את היישום של הפונקציה ()Repeat בשורות 144-155 :

```
144: <xsl:script>
145: <![CDATA[
146: // returns a string consisting of n copies of t
147: function Repeat(t, n)
148: {
149:        var r = "";
150:        while (n-- > 0)
151:               r += t;
152:        return r;
153: }
154: ]]>
155: </xsl:script>
```

התסריט חסום על ידי תגיות xsl:script, והתוכן של התסריט מוחזק בתוך בלוק CDATA כך שמנתח תחביר ה-XSL לא ינתח תווים שמורים כמו > בשורה 150. פונקציה זו לא מבצעת דבר מעבר ליצירת מחרוזת של n עותקים (הפרמטר השני : האורך של הרווח) של t (התו המתקבל :).

טאבים

שורות 123-125 מבצעות עבור הטאבים בדיוק את מה ששורות 119-121 ביצעו עבור הרווחים:

```
123: <xsl:template match="tab">
124:         <span><xsl:eval>Repeat(" ", 4)</xsl:eval></span>
125: </xsl:template>
```

שוב אנו קוראים לפונקציה Repeat, הפעם עם ערך קבוע של 4 עבור האורך. כך, כל טאב יהפוך לארבעה תווי רווח קשיחים.

תווים מיוחדים

למסמך ה-XML שלנו יש מספר רכיבי char המייצגים תווים שדורשים טיפול מיוחד ב-HTML ובשפות סימון אחרות. הרכיב char הוא תלוי-שפה ובעל חשיבות סמנטית. בתלוי-שפה, הכוונה היא שהרכיב char אינו ייחודי לשפת סימון מסוימת; זוהי תצורה קנונית לתווים המיוחדים האלו. בחשיבות סמנטית, הכוונה היא שהרכיב char הוא בעל תיאור עצמי. לדוגמה, כשנראה רכיב char עם התכונה type-emdash, מייד ברור מהו, בניגוד למקרה בו היינו רואים רכיב עם הערך &#8218, למשל.

ישנם יתרונות רבים לאחסון רכיבי התווים במסד הנתונים כמו שהם, אבל עכשיו משאנו יוצרים מסמך HTML, אנו חייבים להמר אותם חזרה ליישויות הנדרשות על ידי HTML.

בשורה 129 אנו מבצעים התאמה לכל רכיב char. בשורה 130 אנו מגדירים בלוק choose כדי שנוכל לבצע התאמה בין התכונה לסוג התו שאנו שוקלים:

```
129: <xsl:template match="char">
130:         <xsl:choose>
```

שורות 131-138 מיישמות את הטרנספורמציה של כל הסוגים המיוחדים שידוע לנו עליהם:

```
131:                 <xsl:when test="@type[. = 'emDash']">&#8212;</xsl:when>
132:                 <xsl:when test="@type[. = 'bullet']">&#8226;</xsl:when>
133:                 <xsl:when test="@type[. = 'ellipsis']">&#8230;</xsl:when>
134:                 <!-- we'll just use 2 nbsps for emSpace - #8195 doesn't seem
                 ➥to work as documented -->
135:                 <xsl:when test="@type[. = 'emSpace']">  </xsl:when>
136:                 <xsl:when test="@type[. = 'smartApos']">&#146;</xsl:when>
137:                 <xsl:when test="@type[. = 'smartLQuote']">&#147;</xsl:when>
138:                 <xsl:when test="@type[. = 'smartRQuote']">&#148;</xsl:when>
```

למשל, כשהתו המיוחד הוא מטיפוס bullet, תבוצע הטרנספורמציה בשורה 132; מסמך DOM הפלט יכלול את המחרוזת •, היישות המתאימה לתבליט.

בשורה 139 ניתן לראות את הרכיב xsl:otherwise, שיתאים במידה וכל שאר ההצהרות when לא יתאימו. שוב, מדובר במלכודת בקוד לסימון תווים שלא ציפינו להם. במקרה זה ניצור תגית span עם תכונת המחלקה unknown ; שתכונה הוא המילה char ; מלווה בערך של התכונה type.

המחלקה unknown מציגה את הטקסט באדום, כפי שניתן לראות בשורה 17 :

```
17:          .unknown {color:red;}
```

ההצהרה choose נסגרת בשורה 140, ורכיב התבנית נסגר בשורה 141.

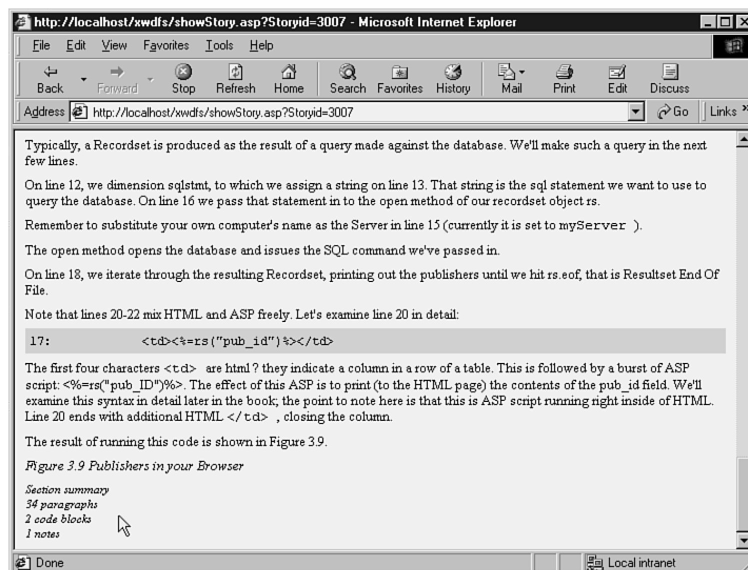
טיפול בהנחיות ייצור

ביצענו התאמה לכל הרכיבים שעל קיומם אנו יודעים, ומעוניינים לעבד אותם, מלבד הנחיות הייצור. אלו הם רכיבים במסמך ה-XML שלנו שנוצרו כדי להכיל הנחיות מיוחדות שיועדו במקור לעורכי Macmillan. אנו לא רוצים להציג אותם בפלט ה-HTML שלנו, אך גם לא רוצים שייפלו ברשתה של מלכודת הרכיבים ה״לא ידועים״ שלנו, לכן בשורות 79-81 נבצע להם התאמה מפורשת אך לא נבצע שום פעולה עבורם :

```
79:  <!-- we ignore PD - these are publishing directions, e.g. "begin note" -->
80:  <xsl:template match="div[@class='PD']">
81:  </xsl:template>
```

סיכום המסמך

בתחתית כל סיפור נמקם סיכום קצר המציין כמה פסקאות ישנן בסיפור, וכמה בלוקי קוד והערות, כמוצג בתרשים 6.8.



תרשים 6.8: סיכום קצר

היישום של סיכום זה מוביל אותנו חזרה לשורות 47-57. אמרנו קודם לכן שכשישנה התאמה ל-story או book, אנו יוצרים את התגית body. בתוך body נמצא ה-HTML המופק על ידי apply-templates (ולכן עיבוד כל הרכיבים האחרים במסמך). לפני שנסגור את התגית <body> בשורה 56 ניצור div חדש עם המחלקה summary:

```
47: <xsl:template match="story|book">
48: <body>
49:     <xsl:apply-templates/>
50:     <div class="summary">
51:         Section summary
52:         <br/><xsl:eval>this.selectNodes("//div").length</xsl:eval> paragraphs
53:         <br/><xsl:eval>this.selectNodes("//code").length</xsl:eval> code blocks
54:         <br/><xsl:eval>this.selectNodes("//note").length</xsl:eval> notes
55:     </div>
56: </body>
57: </xsl:template>
```

הבורר summary. בשורה 27 קובע את סגנון הסיכום לסגנון מוטה בן 8 נקודות ללא שוליים תחתונים:

```
27:     .summary {font-style:italic; font-size:8pt; margin-bottom:0;}
```

אנו מדפיסים את המילים Section summary מלוות בפרטים על מספר הפסקאות, בלוקי הקוד, וההערות שנמצאו. למען האמת, יש בכך שימוש מועט, אך זה מדגים כיצד לגשת ולהציג נתוני-על אודות המסמכים המאוחסנים במסד הנתונים.

אנו מחשבים מספרים אלו באמצעות השיטה selectNodes. שימו לב כי השיטה חייבת להיקרא על שם הרכיב, במקרה זה אנו קוראים לה על שם הרכיב הנוכחי, באמצעות שימוש במילת המפתח this.

כאן, הרכיב הנוכחי הוא הרכיב ברמה העליונה: story או book. בשורה 52 אנו מעבירים כפרמטר ל-selectNodes את תבנית ה-XSL "//div". כך נמצא כל רכיב div שהוא צאצא של הרכיב הנוכחי, ובמלים אחרות, כל div בתוך book או story. מה שנקבל הוא אוסף, ואנו ניגש למאפיין length של אוסף זה אשר יאמר לנו כמה אובייקטים (למשל, כמה רכיבי <div>) ישנם באוסף. לבסוף, אנו מדפיסים ערך זה על ידי שימוש ברכיב <xsl:eval>.

<xsl:eval> מוסר את התוכן שלו למנוע תסריטים, ומחזיר את התוצאה כמחרוזת, המוכנסת במסמך DOM הפלט.

הצעדים הבאים

חילקנו כל פרק לסיפורים, אחסנו את הסיפורים הללו במסד הנתונים, ואחר כך גם הכנו את הסיפורים לתצוגה ב-HTML. בכך בעצם השלמנו את חלקו השמאלי של היישום הסופי אליו אנו שואפים.הצעד הבא שלנו יהיה לבנות טבלת תוכן עניינים שתשרת אותנו במסגרת השמאלית של היישום.

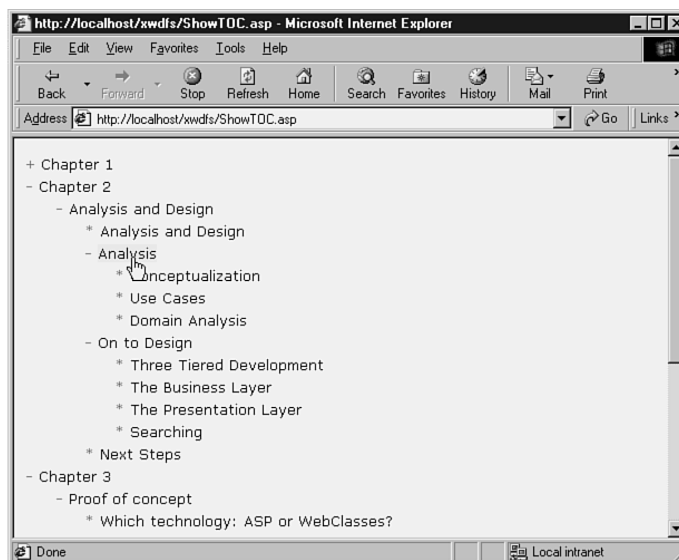
פרק 7
יצירת רכיבים על ידי שימוש ב-XML ו-XSL עם DHTML

בפרק זה:

- אסטרטגיה
- ShowTOC.ASP
- הפיכת ה-XML לטבלת תוכן העניינים
- יצירת ה-HTML מ-XML על ידי שימוש ב-XSL
- הצעדים הבאים

עד כה ראינו כיצד להפוך מסמך HTML למסמך XHTML. אח״כ השתמשנו הן במודל אובייקט המסמך של ה-XML והן ב-XSL כדי ליצור מסמך XML בו ביצענו שינויים ואחסנו אותו במסד הנתונים. בפרק זה נחזור למסד הנתונים, נחלץ פיסות של נתונים, וניצור טבלת תוכן עניינים דינמית מתקפלת, כפי שניתן לראות בתרשים 7.1.

נשים לב לסימנים (+) ו-(-) לצד הנושאים. אלו שעם הסימן + מקופלים; לחיצה על סימני הפלוס תפתח את הנושאים שתחתיהם, והם ייראו כמו אלו עם סימני המינוס. ניתן לראות שהכותרת Chapter 1 מקופלת, בעוד שהכותרת chapter 2 פרושה. פריטים עם כוכבית (*) הם עלים (כלומר, נושאים ללא בנים).



תרשים 7.1: טבלת תוכן העניינים

אנו נבחן את הקוד המשמש ליצירת כל זה בפירוט. אחר כך נבחן מספר דרכים אחרות המשיגות את אותה המטרה.

אסטרטגיה

הפתרון הראשון שלנו לנושא זה, יהיה לחלץ נתונים ממסד הנתונים כדי ליצור מסמך XML DOM פשוט, המשקף את המבנה המקורי של הקטעים, ה-sections, כפי שתואר בפרקים הקודמים. אחר כך נעביר את מסמך ה-XML דרך גיליון סגנונות XSL אשר ייצא קובץ HTML, שנוכל להציג ע״ג הדפדפן. קובץ ה-HTML יכלול קוד JavaScript אשר יהיה אחראי על יצירת טבלת תוכן העניינים הדינמית.

נזכיר כי קטע section ברמה-A מכיל קטע ברמה-B, אשר עשוי להכיל (בין היתר) מספר כלשהו של קטעים ברמה-C, אשר בעצמם עשויים להכיל מספר כלשהו של קטעים ברמה-D.

במסד הנתונים שלנו הפכנו את המבנה הזה לשטוח, אך שמרנו מספיק מידע כדי לשחזר אותו, וזה בדיוק מה שנעשה עכשיו. ברגע שיש לנו מסמך XML DOM עם המבנה הזה, אנו נציג אותו כדף HTML על ידי טרנספורמציית-XSL.

נזכיר כי במסד הנתונים שלנו יש רק טבלה אחת, ולה יש רק מספר שדות, כמוצג בתרשים 7.2.

2:Data in Table 'Stories'

StoryId	ParentId	SectionLevel	Title	XML	TaglessText
2000	0	A	Chapter 2	<?xml version="1.(	Chapter 2
2001	2000	B	Analysis and Desigr	<Long Text>	Analysis and Desigr
2002	2001	C	Analysis and Desigr	<Long Text>	<Long Text>
2003	2001	C	Analysis	<?xml version="1.(	Analysis
2004	2003	D	Conceptualization	<?xml version="1.(	Conceptualization
2005	2003	D	Use Cases	<Long Text>	<Long Text>
2006	2003	D	Domain Analysis	<Long Text>	<Long Text>
2007	2001	C	On to Design	<Long Text>	On to Design
2008	2007	D	Three Tiered Devel	<?xml version="1.(	Three Tiered Devel
2009	2007	D	The Business Layer	<Long Text>	<Long Text>
2010	2007	D	The Presentation L	<Long Text>	<Long Text>
2011	2007	D	Searching	<Long Text>	<Long Text>
2012	2001	C	Next Steps	<?xml version="1.(	Next Steps
3000	0	A	Chapter 3	<?xml version="1.(	Chapter 3
3001	3000	B	Proof of concept	<?xml version="1.(	Proof of concept
3002	3001	C	Which technology:	<Long Text>	<Long Text>

תרשים 7.2: טבלת הסיפורים

לכל רשומה יש StoryID המזהה באופן ייחודי את הסיפור, ו-ParentID, שהוא ה-StoryID של הקטע המכיל אותו. כך, למשל, בתרשים 7.2 ניתן לראות כי לרשומה המודגשת עבור סיפור 2011 (קטע ברמה-D) יש את ההורה 2007. אם נסתכל ארבע שורות למעלה, נמצא ש- 2007 הוא קטע ברמה-C. זה הגיוני; רכיב ברמה-D אמור להיות מוכל בתוך קטע ברמה-C. בטבלת תוכן העניינים שלנו נרצה להזיח (להרחיק מהשוליים) את הקטע הזה ברמה-D בתוך הקטע המכיל אותו ברמה-C.

לסיפור 2007 בעצמו יש ParentID עם הערך 2001. את 2001 אנו מוצאים בשורה השנייה. השדה SectionLevel מגלה שזהו רכיב ברמה-B שה-ParentID שלו הוא 2000.

2000 הוא רכיב ברמה-A, שכזכור הוא פרק. לכל הכניסות ברמה-A יש ParentID עם הערך 0.

יצירת ההיררכיה מחדש

עם ידיעת המבנה הזה, נחלץ כל רשומה ונבנה מחדש את היחסים המקוריים בין הקטעים. למסמך זה בכל אופן, אנו זקוקים רק למידע במידה המספקת, ליצירת טבלת תוכן העניינים. הסיפור בפועל עצמו, כמו גם ההערות, הקוד, וכן הלאה אינם נדרשים לנו (עדיין).

אי לכך, ניצור רכיב חדש, listing, שיחזיק את המידע שאנו צריכים: StoryID, הרמה והכותרת שנציג. ברגע שניצור מסמך DOM המשקף את היחסים הללו, אנו יכולים להציג אותו בדפדפן שלנו באמצעות שימוש ב-XSL לבניית דף HTML.

ShowTOC.ASP

דף ה-ASP שאחראי לסריקת מסד הנתונים ויצירת טבלת תוכן העניינים הוא ShowTOC.ASP, כמוצג בתדפיס 7.1.



השתמשנו במילה TOC בשם קובץ ה-ASP. TOC הינו ראשי תיבות של Table Of Contents - ובעברית "תוכן עניינים". כפי שמשתמע משמו, קובץ זה תפקידו להציג על גבי הדפדפן את תוכן העניינים של הספר. השיטה היא על ידי יצירת קובץ XML מתוך מסד הנתונים, ואחר כך, באמצעות טרנספורמציית XSL, המרת ה-XML ל-HTML.

תדפיס 7.1

```
<!-- #include file ="include.asp" -->
<%
   'ShowTOC.asp - creates the XML for the TOC from the database
   'then converts it into HTML via an xsl stylesheet and displays the results
   dim oXSL, tocXML

   'create the toc - beginning at the root (storyId = 0)
   tocXML =  "<?xml version='1.0'?>" & vbCrLf _
                 & "<toc>" & vbCrLf  & DoTOCSection(0) & "</toc>"

   'transform to HTML via the XSL stylesheet
   set oXSL = Server.CreateObject("FromScratch.XSLTransform")
   oXSL.InputXML = tocXML
   oXSL.XSLFile = fso.BuildPath(codeDir, "Toc2HTML.xsl")
   oXSL.Transform

   'display the HTML
   response.write oXSL.OutputXML


   Function DoTOCSection(id)
        'handle one section element for MakeTOC
        dim s, rs

        'get all the children of the given storyId
        set rs = DBConn.Execute("select StoryId, SectionLevel, Title from Stories
        ➥where ParentId = " & id & " order by StoryId")
        do until rs.eof
             'for each one, create an XML element that contains the particulars
             s = s & "<listing id=""" & rs("storyId") & """
             ➥level=""" & rs("SectionLevel") & """>"  & vbCrLf
             s = s & "<title>" & rs("Title") & "</title>" & vbCrLf
```

```
30:             'recurse into my children
31:             s = s & DoTOCSection(rs("StoryId"))
32:             s = s & "</listing>" & vbCrLf
33:             rs.MoveNext
34:         loop
35:         rs.Close
36:
37:         'return the resulting string
38:         DoTOCSection = s
39:     End Function
40: %>
```

הפעם אנו יוצרים מסמך XML בדרך שונה מכל הדרכים שראינו עד כה. בעבר יצרנו את מסמך ה-DOM שלנו מקובץ, או על ידי שינוי של מסמך DOM אחר, או שיצרנו אותו על ידי שימוש ב-XSL. הפעם, כל העבודה מבוצעת בקובץ ה-ASP. אנו נקרא נתונים ממסד הנתונים, נבנה את המחרוזת בזיכרון, ומזה ניצור את מסמך ה-DOM.

בשורה 0 נכלל include.asp, כפי שראינו בעבר. שורה 1 מתחילה את חלק התסריט של קובץ ה-ASP. העבודה הממשית מתחילה בשורה 7, היכן של-tocXML משויכת מחרוזת:

```
 7:  tocXML =  "<?xml version='1.0'?>" & vbCrLf _
 8:              & "<toc>" & vbCrLf  & DoTOCSection(0) & "</toc>"
```

tocXML הוא מחרוזת שממנה ניצור את מסמך ה-XML DOM כפי שניתן לראות בשורות 11 ו- 12:

```
11:  set oXSL = Server.CreateObject("FromScratch.XSLTransform")
12:  oXSL.InputXML = tocXML
```

בחזרה לשורה 7, ניתן לראות כי הטקסט הראשון במחרוזת הוא הכותרת הסטנדרטית שלנו למסמך XML: הוראת העיבוד <?XML version='1.0'?>, היא כפי שהיינו מצפים. היא מלווה ב-vbCrLf, המחייב תו של שורה חדשה בקובץ, ובכך משאיר את הוראת העיבוד לבד בשורה הראשונה של קובץ ה-XML שלנו.

בהמשך המחרוזת נמצאת התגית <toc>, אשר שוב נקבעת לה שורה משלה. זה מלווה בקריאה לשיטה DoTOCSection, אליה נחזור בעוד רגע. לבסוף, התגית <toc> נסגרת והמחרוזת מסתיימת. זה הכל. ממחרוזת זו ניצור את מסמך הקלט.

כמובן שהעבודה של יצירת גוף מסמך ה-XML נמצאת בשיטה DoTOCSections, אותה נמצא בשורות 20-39.

נשים לב שכשאנו קוראים לפונקציה זו, אנו מעבירים את הערך 0. בשורה 20, הפרמטר נקרא id. נזכיר כי כשיצרנו את הסיפורים, ושמרנו אותם במסד הנתונים, לכל קטע ברמה העליונה ביותר (התגית ברמה-A) שויך parentID עם הערך 0. זו אינה מקריות.

בשורה 25 אנו סורקים את מסד הנתונים. בהינתן storyID, אנו יוצרים אוסף של סיפורים, והם הבנים של אותו ID:

```
25:         set rs = DBConn.Execute("select StoryId, SectionLevel, Title from Stories
            ➥where ParentId = " & id & " order by StoryId")
```

הבה נעיף מבט נוסף ברשומות שבמסד הנתונים שלנו, כמוצג בתרשים 7.3.

2:Data in Table 'Stories'

StoryId	ParentId	SectionLevel	Title	XML	TaglessText
2000	0	A	Chapter 2	<?xml version="1.(	Chapter 2
2001	2000	B	Analysis and Desigr	<Long Text>	Analysis and Desigr
2002	2001	C	Analysis and Desigr	<Long Text>	<Long Text>
2003	2001	C	Analysis	<?xml version="1.(	Analysis
2004	2003	D	Conceptualization	<?xml version="1.(	Conceptualization
2005	2003	D	Use Cases	<Long Text>	<Long Text>
2006	2003	D	Domain Analysis	<Long Text>	<Long Text>
2007	2001	C	On to Design	<Long Text>	On to Design
2008	2007	D	Three Tiered Devel	<?xml version="1.(	Three Tiered Devel
2009	2007	D	The Business Layer	<Long Text>	<Long Text>
2010	2007	D	The Presentation L	<Long Text>	<Long Text>
2011	2007	D	Searching	<Long Text>	<Long Text>
2012	2001	C	Next Steps	<?xml version="1.(	Next Steps
3000	0	A	Chapter 3	<?xml version="1.(	Chapter 3
3001	3000	B	Proof of concept	<?xml version="1.(	Proof of concept
3002	3001	C	Which technology:	<Long Text>	<Long Text>

תרשים 7.3: טבלת הסיפורים, שוב.

ניתן לראות שלכל רשומה יש storyID, ולכל רשומה גם מצביעה ל-ParentID (ה-storyID של הקטע המכיל אותו), פרט לסיפורים ברמה-A, להם יש ParentID עם הערך 0.

כעת נפתח את מנתח השאילתות (Query Analyzer) ונבצע את אותו חיפוש כמו זה שביצענו בדף ה-ASP, כשאנו מתחילים, כפי שעשינו בשורה 7, עם ParentID עם הערך 0 (אפס):

```
Select StoryiID, SectionLevel, Title from Stories where
➥ParentID = 0 order by StoryID
```

תרשים 7.4 מציג את תוצאות הפעלת השאילתה עבור מסד הנתונים לאחר הוספת סיפורים משלושת הפרקים הראשונים.

SQL Server Query Analyzer - [Query - MERCURY.XWDFS.sa - (untitled) - select StoryID,...*]

File Edit View Query Window Help

DB: XWDFS

```
select StoryID, SectionLevel, Title from Stories where ParentID = 0 order by StoryID

StoryID     SectionLevel Title
----------- ------------ ----------------------------------------
1000        A            Chapter 1
2000        A            Chapter 2
3000        A            Chapter 3

(3 row(s) affected)
```

Results

Query batch completed. | Exec time: 0:00:00 | 3 rows | Ln 1, Col 1

תרשים 7.4: בחירת הרמה-A.

כפי שניתן לראות, התוצאה היא החזרת הרשומות עבור רמה-A, כפי שהייינו מצפים.

בשורה 26 אנו יוצרים לולאת do until אשר רצה רשומה רשומה עד סוף ה-record set:

```
26:    do until rs.eof
```

בשורה 22 הגדרנו משתנה s שישמש לבניית מחרוזת, אשר בשורה 38 יהיה הערך המוחזר של פונקציה זו. למעשה, כל מה שנשים במחרוזת זו יוכנס למסמך ה-XML שלנו בין התגיות <toc> ו- </toc>.

בשורה 28 אנו מוסיפים למחרוזת s:

```
28:            s = s & "<listing id=""" & rs("storyId") & """
               ➥level=""" & rs("SectionLevel") & """>"  & vbCrLf
```

לתגית <listing> ניתנת התכונה id עם ה-storyID של הרשומה הנוכחית. במקרה של הרשומה הראשונה שלנו, זהו storyID עם הערך 2000. לתגית listing הנוכחית ניתנת גם תכונה שנייה, level, הנקבעת ל-SectionLevel המוחזר מהשאילתה (במקרה זה "A").

בשורה 29, אנו מוסיפים את הכותרת למחרוזת:

```
29:            s = s & "<title>" & rs("Title") & "</title>" & vbCrLf
```

שוב, מכיון שאנו בונים מסמך XML, הכותרת היא בין תגיות <title>, והערך הוא השדה המתקבל מהשאילתה.

בנקודה זו, בפעם הראשונה, s נראה כך:

```
<listing id="1000" level="A">
<title>Chapter 1</title>
```

בשורה 31 אנו מוסיפים ל-s את תוצאת הקריאה הרקורסיבית ל-DoTOCSections, תוך העברת ה-storyID **הנוכחי** (1000). מנגנון הרקורסיה הזה הוא הסוד ליצירת כל טבלת תוכן העניינים.

עכשיו DoTOCSections תסרוק את מסד הנתונים כדי למצוא את כל הרשומות שה-parentID שלהן הוא 2000. חיפוש זה מניב רשומה אחת, עם ID עם הערך 2001 (הקטע ברמה-B).

כשנחזור לשורה 31, s ייראה כך:

```
<listing id="2000" level="A">
<title>Chapter 2</title>
<listing id="2001" level="B">
<title>Analysis and Design</title>
```

נשים לב לכך שכשפתחנו את תדפיס 2001, עדיין לא סגרנו את תדפיס 2000. כך, תדפיס 2001 **מוכל** בתדפיס 2000.

כשאנו מגיעים לעלה, אנו מוכנים לסגור את התדפיסים הפתוחים שלנו. זה משלים את ההכלה, עם תדפיסים ברמה-D המוכלים בתדפיסים ברמה-C, המצויים בתוך רמה-B וכן הלאה.

ניתן לראות את ההשפעה של זה על ידי ביצוע הצעדים הבאים:

1. פתח את ShowTOC.ASP בעורך.
2. עבור לשורה 34 והכנס שורה חדשה מתחת, כך שהקוד ייראה כך:

```
loop
Response.Write(s)
rs.Close
```

3. שמור את הקובץ הזה ופתח את ShowTOC.ASP בדפדפן.
4. הצג את HTML המקור.

התוצאה מוצגת בתרשים 7.5.

כל קטע ברמה-A ימצא את הרמה-B שלו, אשר בתורה תמצא את כל הקטעים ברמה-C שלה, וכל אחד מהם ימצא את כל הקטעים ברמה-D שלו.

הרקורסיה ממשיכה עד שכל רשומה התווספה למחרוזת שלנו. אז, בשורה 35, אנו סוגרים את recordSet ובשורה 38 מחזירים את המחרוזת. זה מוביל אותנו חזרה לשורה 8, שם אנו ממקמים את תגית הסגירה </toc>.

```
ShowTOC[1] - Notepad
File  Edit  Search  Help
<listing id="2000" level="A">
<title>Chapter 2</title>
<listing id="2001" level="B">
<title>Analysis and Design</title>
<listing id="2002" level="C">
<title>Analysis and Design</title>
</listing>
<listing id="2003" level="C">
<title>Analysis</title>
<listing id="2004" level="D">
<title>Conceptualization</title>
</listing>
<listing id="2005" level="D">
<title>Use Cases</title>
</listing>
<listing id="2006" level="D">
<title>Domain Analysis</title>
</listing>
</listing>
<listing id="2007" level="C">
<title>On to Design</title>
<listing id="2008" level="D">
```

תרשים 7.5: תוצאות הפעולות הננקטות ב-ShowTOC.ASP.

בשורה 11 אנו יוצרים את האובייקט FromScratch.xslTransform שראינו קודם לכן, ובשורה 12 אנו קובעים את המאפיין InputXML שלו למחרוזת שזה עתה יצרנו. זה קורא ל-ParseIntoDOM, כפי שראינו בפרקים קודמים, ונוצר מסמך XML DOM חדש המכיל את ה-XML מהמחרוזת ב-tocXML.

הפיכת ה-XML לטבלת תוכן העניינים

עכשיו כשיש לנו את מסמך ה-XML שלנו, אנו יכולים להתחיל בתהליך יצירת טבלת תוכן עניינים לתצוגה. כדי להשיג זאת, נשתמש ב-XSL, שיהפוך את ה-XML שלנו ל-HTML, ובדרך נכתוב את התסריט בצד הלקוח.

לפני שנוכל לקחת על עצמנו את השימוש ב-XSL להפיכת ה-XML שלנו ל-HTML, אנו חייבים לדעת כיצד ייראה ה-HTML. בניית טבלת תוכן עניינים (TOC) מתקפלת, אינה מטלה רגילה, כך שנתחיל בכך שהיא תפעל ב-HTML. ברגע שנדע איזה HTML ותסריטים אנו צריכים, יהיה קל יותר ליצור את ה-XSL.

בניית ה-TOC ב-HTML

ה-TOC המתקפל יכיל את רשימת הכותרות של כל ״סיפור״. בתחילת הטקסט נשרשר אחד משלושה סמלים: סימן פלוס במידה ויש לו בנים והוא מקופל, סימן מינוס אם יש לו בנים והוא פרוש, וכוכבית אם הוא עלה. ראינו זאת בתרשים 7.1.

כדי להתחיל בכך, ניצור את התסריטים לשינוי התדפיסים. ברגע שזה יעבוד, נהיה מסוגלים לחזור ל-XSL וליצור את התסריטים, ה-HTML, והנתונים ממסמך ה-DOM XML שנוצר ממסד הנתונים.

תדפיס 7.2 מציג את ה-HTML הדרוש להשגת מטרת העיצוב.

תדפיס 7.2

```
<html>
  <head>
  <style>
        * {font-family:Verdana; font-size:10pt;}
        .listing {margin-bottom:3pt; cursor:hand;}
        .expand {color:red;}
  </style>
  <script>
  <!--
  function Expand()
  {
        // get to the "owning" div
        var e = window.event.srcElement.parentElement.parentElement;

        var sign = e.children(0).children(0);        // the plus/minus sign span

        // if we were expanded, then hide the contents
        // don't do anything if there's no sign (ie. leaf node)
        switch (sign.innerText)
        {
```

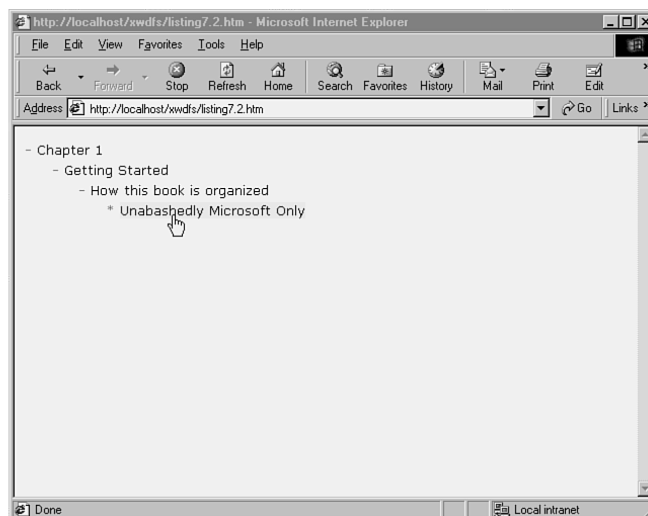
```
        case '-':
                sign.innerText = '+';
          for (var i = 1; i < e.children.length; i++)
            e.children(i).style.display="none";
                break;
        case '+':
                sign.innerText = '-';
          for (var i = 1; i < e.children.length; i++)
            e.children(i).style.display="";
                break;
        }
}
function Highlight(on)
{
        // change the background as the mouse passes over selections
        event.srcElement.style.backgroundColor = (on ? "yellow" : "");
        return false;
}
function ShowStory()
{
        // cause the rh frame to display the selected story - we have the story id
        ➥as the listing element's id
        var srcId = event.srcElement.id;

        // tack on options to control the "mode" of display, based on checkboxes
        ➥in BookControls.htm
        var NonCSS = parent.frames.bottom.NonCSS.checked;
        var CodeBlocks = parent.frames.bottom.CodeBlocks.checked;

        parent.frames.rh.location.href = "ShowStory.asp?StoryId="
        ➥+ srcId + (NonCSS ? "&NonCSS=y" : "") +
        ➥(CodeBlocks ? "&CodeBlocks=y" : "");
        return false;
}
-->
</script>
</head>
<body>
        <div>
                <div class="listing" style="margin-left:0em">
                        <span class="expand" onclick="Expand()">-</span>
                        <span onmouseover="Highlight(true)"
                                onmouseout="Highlight(false)"
                                onclick="ShowStory()" id="1000">Chapter 1</span>
```

```
            </div>
            <div>
                <div class="listing" style="margin-left:2em">
                    <span class="expand" onclick="Expand()">-</span>
                    <span onmouseover="Highlight(true)"
                    onmouseout="Highlight(false)"
                    onclick="ShowStory()" id="1001">
                    ➥Getting Started</span>
                </div>
                <div>
                <div class="listing" style="margin-left:4em">
                <span class="expand" onclick="Expand()">-</span>
                <span onmouseover="Highlight(true)"
                    onmouseout="Highlight(false)"
                    onclick="ShowStory()" id="1002">How this book
                    ➥is organized</span>
                </div>
                <div>
                <div class="listing" style="margin-left:6em">
                <span class="expand" onclick="Expand()">*</span>
                <span onmouseover="Highlight(true)"
                onmouseout="Highlight(false)"
                onclick="ShowStory()" id="1003">Unabashedly
                ➥Microsoft Only</span>
                </div>
            </div>
        </div>
  </body>
</html>
```

אם נשים את הקוד הזה בקובץ (ללא מספרי השורות, כמובן) ונשמור אותו כקובץ HTML, נוכל להציג אותו בדפדפן ולראות שהתוכן אכן ניתן לפרישה ולקיפול, כמוצג בתרשים 7.6.



תרשים 7.6: פרישה וקיפול של התוכן.



כדאי לנסות! פתח מסמך XML בתוך דפדפן explorer 5, ללא שימוש בטרנספורמציות XSL, גיליונות סגנון או מסמך ASP. שים לב כיצד יוצג מסמך ה-XML בדפדפן. מראה מתקפל זה, הוא ברירת המחדל של explorer 5 להצגת מסמכי XML. ללא כל קשר, מראה זה מזכיר את המראה המתקפל של ה-TOC שלנו.

כדי לאפשר לקסם הזה לפעול, אנו זקוקים לשלושה תסריטים בצד הלקוח, המוצגים בשורות 9-50. אנו נבחן אותם בסדר מורכבות עולה, כשנתחיל עם הפשוט ביותר: Highlight().

Highlight

Highlight, המוצג בשורות 31-36, לא עושה דבר מלבד לגשת לרכיב שעורר את הליך הטיפול באירועים, ולקבוע את צבע הרקע שלו לצהוב, אם הפרמטר שהועבר הוא "on" או אחרת למחרוזת הריקה (ולכן צבע הרקע לפי ברירת המחדל).

צבע הרקע שבברירת המחדל נקבע ברכיב הסגנון בתוך הרכיב head, כמוצג בשורות 2-6. שורה 3 מראה כי סגנון הטקסט של ברירת המחדל הוא גופן Verdena בן 10 נקודות:

```
3:         * {font-family:Verdana; font-size:10pt;}
```

ניתן לראות, לדוגמה בשורות 56-57, ש-Highlight נקבע כהליך לטיפול באירועים mouseover ו-mouseout עבור הכותרת:

```
56:              <span onmouseover="Highlight(true)"
57:                    onmouseout="Highlight(false)"
```

הדבר חוזר עבור כל כותרת, כפי שניתן לראות בשורות 64, 65, 70, 71, 77, ו- 78.

אנו קוראים ל-highlight כשהעכבר מעל הכותרת, וזה קובע את צבע הרקע לצהוב, ומשחזר אותו לברירת המחדל כשמתעורר האירוע mouseout.

(e) Show Story

מעט יותר מורכב הוא הליך הטיפול באירועים ()ShowStory, הנקרא כאשר המשתמש לוחץ על כותרת מסוימת, כפי שניתן לראות למשל בשורה 58:

```
58:    onclick="ShowStory()" id="1000">Chapter 1</span>
```

הדבר מעורר את השיטה ShowStory המוצגת בשורות 37-48. לכל כותרת יש id, ובשורה 40 למשל, אנו ניגשים ל-ID הזה. אנו עושים זאת על ידי בקשת ה-id של ה-srcElement (זהו רכיב המקור) של המאורע:

```
40:    var srcId = event.srcElement.id;
```

לעת עתה נדלג על שורות 42-44 ונחזור אליהן מאוחר יותר, בהמשך הפרק.

בשורה 46 אנו קובעים את המסגרת הימנית ל-ShowStory.asp, אליו אנו מעבירים את ה-id כפרמטר:

```
46:    parent.frames.rh.location.href = "ShowStory.asp?StoryId="
       ➥+ srcId + (NonCSS ? "&NonCSS=y" : "") +
       ➥(CodeBlocks ? "&CodeBlocks=y" : "");
```

(e) Expand

Expand, המוצגת בשורות 9-30, היא הלב והנשמה של טבלת תוכן העניינים המתקפלת. בשורה 55, למשל, אנו משייכים את ההליך של הטיפול באירוע, לאירוע onclick של סימן המינוס שלנו:

```
55:    <span class="expand" onclick="Expand()">-</span>
```

כל כותרת בדף ה-HTML שלנו תהיה בתוך שני רכיבי div: חיצוני ופנימי. הרכיב div הפנימי יהיה מהמחלקה listing ויכיל מידע על התצוגה כמו כמה רחוק להזיח. הרכיב div החיצוני יהיה נקודת השליטה עבור קיפול ופרישת הכותרת.

כדי לגרום לזה לעבוד, כשנלחץ על סימן מינוס של כותרת מסוימת, אנו חייבים להשיג גישה לרכיב div החיצוני של הכותרת הזו. בשורה 12 אנו עושים זאת על ידי השגת הסבא של ה-srcElement:

```
12:  var e = window.event.srcElement.parentElement.parentElement;
```

הבה נפרק זאת לחלקים. אנו מקבלים אירוע שאפשר להתייחס אליו כ-window.event. אנו יכולים לפנות לרכיב שעורר את האירוע (כאן, הקליק על סימן המינוס) על ידי window.evnet.srcElement. נוכל גם לפנות אל ההורה של srcElement (שזה רכיב ה-div הפנימי) על ידי window.event.srcElement.parentElement. כך אנו מגיעים לרכיב ה-div החיצוני שלו על ידי window.event.srcElement.parentElement.parentElement.

אנו משייכים את הרכיב outerdiv הזה למשתנה המקומי e. אחר כך אנו מבקשים מ-e את הבן הראשון של הבן הראשון שלו (נזכיר כי ב-JavaScript אוספים הם

מבוססי-אפס; הרכיב הראשון הוא רכיב 0). הבן הראשון של הבן הראשון שלו הוא הסימן מינוס (או פלוס), ואנו משייכים אותו למשתנה sign:

```
13:          var sign = e.children(0).children(0);        // the plus/minus sign span
```

הקוד הזה מסתמך על הבנה פנימית של ה-DOM שיצרנו, אך זה בטוח כי זהו אותו הקוד שיצר את ה-DOM, וכך אנו יודעים היכן למצוא את סימן המינוס או הפלוס.
ב-DHTML הטקסט הפנימי של רכיב הוא הטקסט שהוצג בפועל עבור אותו רכיב. במקרה של sign.innerText זה יהיה הסימן '-' או '+' עצמו. בשורה 17, אנו בוחנים את הערך (switch) ונוקטים פעולה בהתאם לכך.



השימוש כאן ב-switch במקום ב-xsl:choose עשוי להיות מבלבל. נזכיר כי בנקודה זו אנו ב-JavaScript שבצד הלקוח, אשר יישלח לדפדפן ויעובד שם.

אם יש לנו סימן מינוס, כפי שקורה בשורות 19-23, נשנה אותו לסימן פלוס (innerText הוא קריאה/כתיבה ב-DHTML, ולכן דרוש IE4 ומעלה, כדי להשתמש ביישום הזה!). ונעבור באיטרציה על כל הצאצאים של רכיב זה ונקבע את מאפיין display שלהם ל-"none".

יצירת ה-HTML מ-XML על ידי שימוש ב-XSL

עכשיו כשה-HTML שלנו עובד, אנו רוצים ליצור קובץ XSL שיפלוט את ה-HTML שאנו צריכים מה-XML שבידינו. נקרא לקובץ זה TOC2HTML.xsl, את הקריאה לו, ראינו בשורה 13 של ShowTOC.ASP:

```
13:          oxsl: xslFile = fso.BuildPath(codeDir, "Toc2HTML.xsl")
```

תוצאת ההחלה של קובץ ה-XSL הזה על קובץ ה-XML שבנינו ממסד הנתונים צריכה להיות מאוד דומה ל-HTML שזה עתה בחנו. הבה נבחן את קובץ ה-XSL בפירוט, כמוצג בתדפיס 7.3.

תדפיס 7.3

```
<?xml version="1.0"?>
<!-- we use this style sheet to display the table of contents -->
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/TR/WD-xsl"
  xmlns="http://www/w3/org/TR/REC-html40"
  result-ns="">

  <!-- default rules -->
  <xsl:template match="text()"><xsl:value-of/></xsl:template>

  <xsl:template match="/">
        <xsl:apply-templates />
```

```
</xsl:template>

<!-- Our base element. Create the HTML structure, including the scripts
➥needed to expand/contract the elements -->
<xsl:template match="toc">
<html>
<head>
<style>
	* {font-family:Verdana; font-size:10pt;}
	.listing {margin-bottom:3pt; cursor:hand;}
	.expand {color:red;}
</style>
<script><xsl:comment><![CDATA[
function Expand()
{
	var e = window.event.srcElement.parentElement.parentElement;
	➥// get to the "owning" div
	var sign = e.children(0).children(0);		// the plus/minus sign span

	// if we were expanded, then hide the contents
	// don't do anything if there's no sign (ie. leaf node)
	switch (sign.innerText)
	{
	case '-':
		sign.innerText = '+';
	  for (var i = 1; i < e.children.length; i++)
	    e.children(i).style.display="none";
		break;
	case '+':
		sign.innerText = '-';
	  for (var i = 1; i < e.children.length; i++)
	    e.children(i).style.display="";
		break;
	}
}
function Highlight(on)
{
	event.srcElement.style.backgroundColor = (on ? "yellow" : "");
	return false;
}
function ShowStory()
{
	var srcId = event.srcElement.id;
	var UseCSS = parent.frames.bottom.UseCSS.checked;
```

```
        parent.frames.rh.location.href = "ShowStory.asp?StoryId=" + srcId
        ➥+ (UseCSS ? "" : "&NonCSS=y");
        return false;
}
]]></xsl:comment></script>
</head>
<body>
        <xsl:apply-templates />
</body>
</html>
</xsl:template>

<xsl:script>
        // global for maintaining the current indent level
        var indent = 0;
</xsl:script>

<!-- Each story has a listing element, that are nested based on the book structure -->
<xsl:template match="listing">
        <div>
                <!-- This inner div controls the indenting level -->
                <div class="listing">
                        <xsl:attribute name="style">margin-left:<xsl:eval>
                        ➥indent</xsl:eval>em</xsl:attribute>
                <!-- The region that contains the +- sign for expanding -->
                <span class="expand" onclick="Expand()">
                        <xsl:choose>
                                <!--
                                        If I have any listing nodes as children,
                                        ➥then I am not a leaf, and so output
                                        ➥a minus sign
                                        for the expand/contract control. If I am a
                                        ➥leaf, then just use a small dot
                                -->
                                <xsl:when test="listing">-</xsl:when>
                                <xsl:otherwise>&#183;</xsl:otherwise>
                        </xsl:choose></span>
                <!--
                        This span holds the story title, and has mouse events to
                        ➥color it as the mouse moves over it
                        Clicking in this span will display the story in the rh pane,
                        ➥which is why we need the id attrib
                -->
```

```
 90:            <span onmouseover="Highlight(true)" onmouseout="Highlight(false)"
                ➥onclick="ShowStory()">
 91:                <xsl:attribute name="id"><xsl:value-of select=
                    ➥"@id"/></xsl:attribute>
 92:                <xsl:value-of select="./title"/></span>
 93:            </div>
 94:            <!--
 95:                bump up the indent before we recurse, then restore it
                    ➥when we're done
 96:                we have to use a global since there's no good way to pass
                    ➥in an argument
 97:            -->
 98:            <xsl:eval>indent += 2;"";</xsl:eval>
 99:            <xsl:apply-templates />
100:            <xsl:eval>indent -= 2;"";</xsl:eval>
101:        </div>
102:    </xsl:template>
103:
104:    <!-- We don't have to do anything with titles, since we retrieved its value
        ➥when processing the listing element -->
105:    <xsl:template match="title">
106:    </xsl:template>
107:
108: </xsl:stylesheet>
```

כעת אתם צריכים כבר להרגיש נוח בסריקת מסמך XSL, ולכן לא נסקור בפירוט את אותו החומר שבחנו מוקדם יותר בספר. בשורה 0 ניתן לראות, למשל, את הוראת העיבוד הסטנדרטית, ובשורות 2-5 אנו יוצרים את ה-namespaces. שורה 8 מטפלת ברכיבי טקסט, ושורות 10-12 מבצעות התאמה לרכיב השורש ועוברות ברקורסיה על מסמך DOM הקלט שב-XML.

הרכיב החדש הראשון שנראה, הוא בשורה 15, היכן שאנו מבצעים התאמה לרכיב toc שיצרנו בשורה 8 בתדפיס 7.1:

```
15:   <xsl:template match="toc">
```

בשורות 16-63 נכתב קוד HTML. התוצאה היא שהרכיב TOC מוחלף על ידי קוד HTML (בראש מסמך ה-HTML של טבלת תוכן העניינים שלנו). הבה נבחן זאת ביתר פירוט.

שורות 17 ו-18 יוצרות את התגיות הצפויות html ו-head. שורות 18-22 מגדירות שלושה סגנונות. הראשון, המוצג בשורה 19, יהיה סגנון ברירת המחדל עבור מסמך ה-HTML החדש שלנו. השני, בשורה 20, יהיה הסגנון המשויך לכל רכיב המסומן עם התכונה "class="listing. לבסוף, בשורה 21, אנו יוצרים סגנון הקרוי expand עבור סימני הפלוס והמינוס.

שורה 23 יוצרת רכיב script במסמך DOM הפלט. הוא ייצור תסריט בצד הלקוח בקובץ ה-HTML, אשר יורץ בדפדפן של המשתמש:

```
23:    <script><xsl:comment><![CDATA[
```

הערות XSL אינן מועברות הלאה לדף ה-HTML. אם אנו רוצים הערה ב-HTML (ואנו רוצים) אנו חייבים להשתמש ברכיב xsl:comment כדי ליצור אותה. אנו רוצים לשים את התסריט שלנו בתוך הערה בקובץ ה-HTML, כדי שדפדפנים שלא יכולים לטפל בתסריטים, יתעלמו מהם.

בסוף שורה 23 אנו מוסיפים תגית <![CDATA[, האומרת למנתח תחביר ה-XSL שתוכן ההערה אינו מיועד לניתוח (כך, אנו יכולים להשתמש בתווים שמורים כמו הסימנים < ו- >).

בשורות 23-57 נמצא התסריט שבצד הלקוח אשר אמור לטפל בהרחבה וכיווץ רכיבים בטבלת תוכן העניינים. זהו בדיוק התסריט שבחנו בקובץ ה-HTML שלנו, המועבר ישירות למסמך ה-XSL שלנו, ומסומן ב-CDATA כך שלא יעובד, אלא רק יועבר הלאה ל-HTML הפלט.

בשורה 57 אנו סוגרים את הרכיב CDATA, את הרכיב xsl:comment, ואת רכיב תסריט ה-HTML. בשורה 58 אנו סוגרים את הרכיב head של מסמך ה-HTML.

שורות 59-63 מבטיחות שהפלט הנותר יהיה תחום בין תגיות body, ושכל מסמך ה-HTML יסתיים עם תגית </html> סוגרת.

יצירת רכיבי התדפיס

מה שנכנס לתוך התגית body, נקבע על ידי מה שנותר בדף ה-XSL.

שורות 65-68 יוצרות משתנה גלובאלי לשימוש ביתרת המסמך. כדי לעשות כן, אנו פותחים xsl:script, אך העבודה היחידה שאנו עושים בתסריט היא ליצור את המשתנה indent ולאתחל אותו ל- 0 (אפס). נראה למה זה משמש בעוד רגע:

```
65:  <xsl:script>
66:        // global for maintaining the current indent level
67:        var indent = 0;
68:  </xsl:script>
```

נזכיר כי בתדפיס 7.1 יצרנו רכיבי תדפיס. בשורה 71 של תדפיס 7.2 ביצענו התאמה לרכיבי התדפיס האלו:

```
71:    <xsl:template match="listing">
```

לכל תדפיס שנמצא, נוסיף עבור מסמך הפלט רכיב div, אליו נתייחס כאל "הרכיב div החיצוני" ("Outer Div") מכיון שבתוך רכיב זה ניצור רכיב div שני, "פנימי". לרכיב div הפנימי תהיה תכונה class עם הערך listing. התוצאה היא שה-div הפנימי ישתמש בסגנון עבור תדפיס, אותו הוספנו קודם בשורה 20 (לכל תדפיס יש שוליים תחתונים של 3 נקודות, והוא קובע את סמן העכבר לכף היד).

נזכיר שבמסמך ה-HTML שלנו, יצרנו הזחה עבור רכיבים מוכלים על ידי שימוש ברכיב סגנון. אנו רואים זאת למשל בשורות 54 ו-61 בתדפיס 7.2 :

```
54:    <div class="listing" style="margin-left:0em">
61:    <div class="listing" style="margin-left:2em">
```

בשורה 75 של קובץ ה-XSL שלנו, המוצג בתדפיס 7.3, אנו נותנים ל-div הפנימי את התכונה style שערכה יהיה :margin-left מלווה בערך המשתנה הגלובאלי indent. אחר כך אנו מוסיפים את האותיות "em" לערך של התכונה. לכן כרגע הוא ייצור :

```
<div class="listing" style="margin-left:0em">
```

נראה מוכר? זה בדיוק מה שראינו בשורה 54 של ה-HTML.

אחר כך אנו יוצרים רכיב span שיכיל את הפלוס, מינוס, או הכוכבית. ל-span יש תכונת מחלקה expand, ומטפל אירוע (״טריגר״). התוצאה היא שמסמך הפלט מכיל עתה גם את התגית הבאה :

```
<span class="expand" onclick="Expand()">
```

תוכן תגית ה-span מוחלט בשורות 78-85. את הרכיבים xsl:choose, xsl:when ו-xsl:otherwise ראינו קודם לכן.

לכל שורה תינתן כוכבית (*) אם היא עלה, או סימן מינוס (-) אם יש לה בנים. לאף שורה לא יינתן הסימן (+) בשלב זה, מכיון שהמפרט שלנו מציין שכל הנושאים פרושים במלואם כשהם מוצגים לראשונה.

בשורה 83 אנו משתמשים בתכונה test=pattern של הרכיב xsl:when :

```
83:    <xsl:when test="listing">-</xsl:when>
```

הוא מחזיר true אם סריקת הבנים של הרכיב הנוכחי, תמצא לפחות אחד שיתאים לתבנית. התבנית בה אנו משתמשים כאן היא listing ; התוצאה צריכה לקבל התאמה כשלרכיב הנוכחי יש בנים שהם גם תדפיסים (לדוגמה, כאשר לרמה-C יש בנים ברמה-D). אם אכן ישנה התאמה, הפלט יהיה סימן מינוס (-), אחרת זהו עלה והפלט יהיה כוכבית. שורה 85 סוגרת את xsl:choose ואת תגית ה-span.

הפלט שלנו, עבור תדפיסים עם בנים, ייראה עכשיו כך :

```
<span class="expand" onclick="Expand()">-</span>
```

כעת, אנו צריכים ליצור תגית span עבור הכותרת עצמה. הפלט יהיה רכיב span עם שלושה מטפלי אירוע בשורה 90 :

```
90:    <span onmouseover="Highlight(true)" onmouseout="Highlight(false)"
       ➥onclick="ShowStory()">
```

אנו מוסיפים לתגית ה-span, תכונה נוספת, שהיא רכיב של ה-HTML , הקרויה id, וערכה נקבע לערך תכונת רכיב ה-XML הנוכחי id. לכן, תגית ה-span הפותחת בפלט ה-HTML תיראה כך :

```
<span onmouseover="Highlight(true)" onmouseout="Highlight(false)"
➥onclick="ShowStory()" id="1000">
```

לאחר מכן נוציא את פלט הכותרת. אנו משיגים את הכותרת על ידי שימוש ברכיב xsl:value-of עם התכונה select המחפשת אחר הרכיב title, שהוא בן של הרכיב הנוכחי.

לעצם העניין

כפי שאתה ודאי יודע, לעתים קרובות במטלות תכנות, יש יותר מדרך אחת להשגת אותה המטרה. המצב דומה ב-XML. כאן יצרנו את ה-TOC שלנו על ידי חילוץ סיפורים ממסד הנתונים, בניית מסמך XML DOM באמצעות שרשור מחרוזות, והחלת גיליון סגנונות XSL.

נזכיר כי לפני ששמנו את הסיפורים **בתוך** מסד הנתונים, יצרנו קודם קבצי XML לכל פרק. אם היינו רוצים, היינו יכולים ליצור את אותה טבלת תוכן עניינים דינמית על ידי חזרה לאותם פרקים והחלת קובץ XSL עליהם ליצירת ה-TOC.

למעשה, יש לנו שתי אפשרויות. מצד אחד, היינו יכולים לשנות את קובץ ה-XML של הפרק, למסמך DOM של טבלת תוכן עניינים. לא ניתן יהיה להבדיל בינו לבין זה שנוצר ממסד הנתונים. לאחר מכן, היינו פשוט מחילים את אותו קובץ XSL כפי שעשינו בפרק הזה: Toc2HTML.xsl.

מצד שני, אנו יכולים לשנות את ה-XML של הפרק ישירות ל-HTML הנדרש על ידי יצירת קובץ XSL חדש, שניתן לכנותו xml2toc2HTML.xsl.

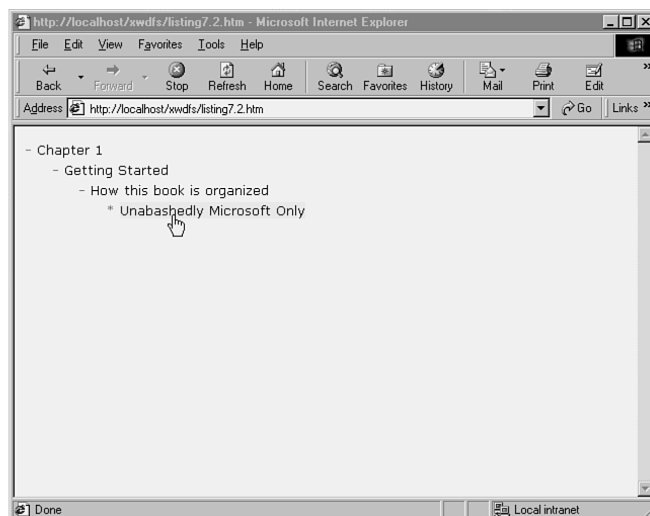
הבה נבחן כל אחת משתי הדרכים, כדי לראות כיצד ניתן לבצע זאת.

מ-XML ל-XML

נתחיל בכתיבת קובץ XSL שנוכל להחיל על chap2.xml ולהפיק קובץ פלט שיוזן ל-toc2html.xls. חשוב לציין, כי קובץ ה-XML שנפיק יהיה זהה למסמך ה-XML DOM שזה עתה יצרנו ממסד הנתונים; הפעם ניצור אותו ישירות מ-chap2.xml.

כדי לעשות זאת, יעזור לחזור על מבנה קובץ XML ה-קלט (chap2.xml) ואז לשקול את המבנה של הקלט הנדרש על ידי Toc2HTML.xsl.

אנו יודעים שה-XML לו מצפה Toc2HTML.xsl הוא פשוט מאוד, כפי שניתן לראות בתרשים 7.7.



תרשים 7.7: תוצאות הפעולות הננקטות על ידי ShowTOC.ASP, שוב.

נזכיר כי המבנה הופק על ידי קריאה ממסד הנתונים, ויצירת מחרוזת ששימשה ליצירת מסמך DOM. מסמך ה-DOM הוזן אחר כך ל-Toc2HTML.xsl.

Chap2.xml הוא הרבה יותר מורכב; להלן קטע מתוך הקובץ בתדפיס 7.4.

תדפיס 7.4

```
<?xml version="1.0"?>
<!DOCTYPE book SYSTEM "canon.dtd">
<book><section level="A" id="2000"><title>Chapter 2</title>
➥<section level="B" id="2001"><title>Analysis and Design</title>
➥<div class="FT">
This is not a book on Object-oriented Analysis and Design. I already wrote
➥one of those (
        <i>
Beginning Object-oriented Analysis and Design. Wrox Press 1998 ISBN1-861-001-33-9)
        </i>

, and it takes a few hundred pages to explain OOA&amp;D in any detail.
      </div>
                <div class="FT">
That said, we can't examine a program in depth without understanding how
➥it will be used, and so we must spend some time on analysis. In addition,
➥before we examine the code, we'll want to understand the object model
➥we plan to implement.
      </div>
                <div class="FT">
```

```
This chapter will provide a whirl-wind description of the analysis and design
➥of EmployeeNet: an application for managing human resources in
➥a middle-sized corporation. Hang on to your hat, we're going to do this very quickly.
    </div>
            <section level="C" id="2002"><title>Analysis and
            ➥Design</title><div class="FT">
With a large project such as EmployeeNet, I would expect to invest
➥a significant amount of time on Analysis (understanding the requirements) and
➥design (modeling the solution) before beginning implementation (writing the code).
    </div>
                    <div class="FT">
In 1992 I would have speculated that EmployeeNet was a two year project.
➥I would have been wrong, of course; by the time we were done, it would have
➥slipped out to three years!
    </div>
                    <div class="FT">
By 1995 I might have known that we had less than a year to build a project
➥like this, because the world is speeding up, and Internet sites had better hit
➥the market in under a year.
    </div>
                    <div class="FT">
The tools are getting better, we're all getting smarter about how to build
➥Web applications and the competition is fierce. The time frame keeps shrinking.
➥Today, I believe we must endeavor to get a first release out the door
➥in 4-6 months. Perhaps less. To do this, I'd spend about a month on analysis,
➥another month on design and then two months coding and a month testing.
➥These estimates make it sound like one phase ends before the other begins
➥which is not true, but they do give a good rough and ready approximation
➥of how I'd divide my time.
    </div>
            </section><section level="C" id="2003"><title>Analysis
            ➥</title><div class="FT">
EmployeeNet will manage the human resources needs of middle-sized corporations.
➥The project is being sponsored by Acme Manufacturing, and we'll build it
➥to meet their needs and then generalize for other companies as we go.
    </div>
                    <div class="FT">
Acme employees 2,000 people in the manufacture of high-end consumer products.
➥Their principal products are the Acme Widget and their world-famous Gizmo.
➥They have manufacturing plants in East Podunk and New Boondock
➥and their main offices are in Gotham.
    </div>
                    <section level="D" id="2004"><title>
                    ➥Conceptualization</title><div class="FT">
```

```
35: EmployeeNet will allow the personnel department to track all benefits
    ➥for employees, and will allow employees to review and edit
    ➥their own employment records.
36:      </div>
37:                          </section><section level="D" id="2005"><title>
                             ➥Use Cases</title><div class="FT">
38: There was a time when the typical requirements for a software project
    ➥were expressed in terms of capabilities and performance. While this ensured
    ➥that the resulting system met certain specified benchmarks,
39:        <spacerun len="1"/>
40:
41: it did not ensure that anyone could or would want to use it. Typically the user
    ➥didn't factor into consideration until after the product was out the door.
42:      </div>
43:                                  <div class="FT">
44: Object-oriented analysis begins with a thorough understanding of how
    ➥the product will be
45:        <i>
46: used
47:        </i>
48: . A
49:         <b>
50: use-case
```

כפי שניתן לראות, אנו זקוקים רק לחלק קטן מאוד של המידע בקלט של קובץ ה-XML, כדי להפיק את הפלט של קובץ ה-XML שלנו. באופן עקרוני, אנו צריכים למצוא את רכיבי הקטעים (המודגשים בקטע שלפנינו) ולהפוך אותם לרכיבי תדפיסים, תוך איסוף הכותרות בדרך, ושמירה על המבנה המקונן. מכל שאר הרכיבים, אפשר להתעלם.

כדי לבצע זאת, נשמור את Control.asp כ-ControlR2.asp (הגהה 2), ונוסיף אפשרות תפריט עבור Make TOC from XML, כפי שניתן לראות בקטע הבא:

```
55:   case "Make TOC from XMl"
56:          Response.Write MakeTOCFromXML()
```

ניתן לראות כי זה מעורר את פונקציית התסריט ()MakeTOCFromXML, המוצגת בתדפיס 7.5 (מספור השורות מציין כי זהו קטע מתוך הקובץ ControlR2.ASP).

תדפיס 7.5

```
180:  Function MakeTOCFromXML()
181:          'convert XML file to TOC Listing file
182:          dim oXSL, fn
183:
184:          set oXSL = Server.CreateObject("FromScratch.XSLTransform")
185:
186:          'convert xhtml to xml via a stylesheet
```

```
187:        oXSL.InputFile = dataPath & ".xml"
188:        oXSL.XSLFile = fso.BuildPath(codeDir, "XML2TOC.xsl")
189:
190:        oXSL.Transform
191:
192:        fn = dataPath & ".toc.xml"
193:        oXSL.SaveOutputAsFile fn
194:
195:        MakeTOCFromXML = "TOC constructed"
196:    End Function
```

הפעולה של קוד זה היא ליצור מופע של אובייקט ה-ActiveX שלנו, המוצג בשורה 184, ולספק כקלט את קובץ ה-XML המקורי של הפרק, המוצג בשורה 187, כמו גם קובץ ה-XSL, המוצג בשורה 188.

אחר כך אנו קוראים ל-transform, המוצגת בשורה 190, שזו **בדיוק** השיטה שקראנו לה בעבר, כדי להחיל דף XSL על מסמך XML DOM.

קובץ הפלט chap2.toc.xml נוצר בשורות 192-193. העבודה האמיתית כמובן, מבוצעת ב-XML2TOC.xsl, כפי שנראה בתדפיס 7.6.

תדפיס 7.6

```
<?xml version="1.0"?>

<!--
  Given a chapter's .xml file, (e.g., chap3.xml) create an output
  xml which can be read by toc2html.xsl
-->

<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/TR/WD-xsl"
  xmlns="http://www/w3/org/TR/REC-xml"
  result-ns="">

  <xsl:template match="/">
        <xsl:pi name="xml">version="1.0"</xsl:pi>
        <xsl:apply-templates />
  </xsl:template>

  <!-- catch any unknown tags -->
  <xsl:template match="*">
        <unknown><xsl:attribute name="tag"><xsl:node-name/></xsl:attribute>
              <xsl:apply-templates />
        </unknown>
  </xsl:template>
```

```
23:
24: <!-- our top level tag -->
25: <xsl:template match="book">
26:         <toc>
27:                 <xsl:apply-templates/>
28:         </toc>
29: </xsl:template>
30:
31: <!-- tags we can ignore -->
32: <xsl:template match="head|title|style|body|script|a|meta|link|div|b|i|spacerun|
    ➥tab|note|u|noteline|codeline|code|char"><xsl:apply-templates/></xsl:template>
33:
34: <!-- find the sections, create listings. Then create titles. -->
35: <xsl:template match="section">
36:         <listing>
37:                 <xsl:attribute name="id"><xsl:value-of select="@id"/></xsl:attribute>
38:                 <xsl:attribute name="level"><xsl:value-of select=
                    ➥"@level"/></xsl:attribute>
39:                 <title><xsl:value-of select="./title"/></title>
40:                 <xsl:apply-templates/>
41:         </listing>
42: </xsl:template>
43: </xsl:stylesheet>
```

בשלב זה, 15 השורות הראשונות אמורות להיות מאוד מוכרות. שורות 18-25 מטפלות בתגיות בלתי מזוהות; אם הכל מתנהל כשורה, הן לעולם לא מתבצעות.

העבודה היא בין שורות 24 ו-42. מטרתנו לבצע התאמה לרכיב book בקובץ ה-XML ולהפוך אותו לתגית toc. אחר כך אנו נבצע התאמה לכל קטע, וניצור את רכיבי התדפיס id, level, ו-title שיהוו את קובץ הפלט שלנו, אותו נוכל להזין ל-Toc2HTML.xsl.

שורות 24-29 מבצעות התאמה לרכיב book (אמור להיות רק אחד כזה), ויוצרות את רכיב toc החדש שלנו. כל מה שנמצא, יהיה תחום בתוך תגית הרכיבים <toc> הפותחת ותגית </toc> הסוגרת, היא תגית השורש שלנו.



ידוע לנו כי איננו צריכים להתאים רכיבי סיפור, מאחר והקלט שלנו הוא תמיד פרק, לדוגמה, chap2.xml. אנו יודעים שפרקים תמיד מורכבים מרכיבי book, כמתואר על ידי ה-DTD שלנו.

בשורה 32 אנו מתעלמים במפורש מכל התגיות שאנו יודעים על קיומן ושאיננו מתעניינים בהן. אנו מתעלמים במיוחד ובמפורש מן התגית title; מכיון שהכותרת תיאסף יחד עם הרכיב section, כפי שניתן לראות בשורות האחרונות של הקובץ. הבה נבחן אותן ביתר פירוט.

בשורה 35 מתקבלת התאמה לרכיב קטע, section, ואנו יוצרים רכיב listing במסמך DOM הפלט. אנו משייכים שתי תכונות לרכיב listing : id ו-level.

שתי התכונות נלקחות ישירות מהתכונות המתאימות ברכיב ה-section. אחר כך אנו מוסיפים רכיב title, וסורקים את הבנים הישירים של הרכיב הנוכחי (section), במטרה למצוא רכיב title תואם ולשייך אותו לרכיב title החדש שלנו.

זה הכל. כשאנו קוראים ל-Transform כפי שניתן לראות בשורה 190 בתדפיס 7.5, קובץ ה-XML שלנו משתנה לפי חוקי טרנספורמציית ה-XSL שיצרנו. לבסוף, הקובץ המתקבל נשמר לדיסק, כמוצג בתדפיס 7.7.

תדפיס 7.7

```
<?xml version="1.0"?>
<toc>
   <listing id="1000" level="A">
      <title>Chapter x1</title>
      <listing id="1001" level="B">
         <title>Getting Started</title>
         <listing id="1002" level="C">
            <title>How this book is organized</title>
            <listing id="1003" level="D">
               <title>Unabashedly Microsoft Only</title>
            </listing>
            <listing id="1004" level="D">
               <title>About the project: EmployeeNet</title>
            </listing>
         </listing>
         <listing id="1005" level="C">
            <title>What tools you need</title>
            <listing id="1006" level="D">
               <title>How many machines?</title>
            </listing>
            <listing id="1007" level="D">
               <title>Setting up your development environment</title>
            </listing>
         </listing>
         <listing id="1008" level="C">
            <title>What do you already need to know?</title>
            <listing id="1009" level="D">
               <title>Scale</title>
            </listing>
         </listing>
         <listing id="1010" level="C">
            <title>Distributed interNet Applications  N-Tier development</title>
            <listing id="1011" level="D">
```

```
                    <title>A brief history</title>
                </listing>
                <listing id="1012" level="D">
                    <title>Logical vs. Physical layers</title>
                </listing>
            </listing>
            <listing id="1013" level="C">
                <title>Components and Microsoft's Component Object Model</title>
                <listing id="1014" level="D">
                    <title>MTS and COM+</title>
                </listing>
            </listing>
            <listing id="1015" level="C">
                <title>Next Steps</title>
            </listing>
        </listing>
    </listing>
</toc>
```

כעת ניתן לצפות בקובץ זה על ידי שימוש **באותו** קובץ Toc2HTML.xsl ששימש אותנו בבניית המבנה ממסד הנתונים.

מ-XML ל-HTML

למה לטרוח ביצירת הקובץ הזמני הזה? לא יותר פשוט להוציא פלט HTML? מספר שינויים לקובץ ה-XSL שלנו ישיגו זאת. ידוע לנו מה נדרש כי כבר כתבנו Toc2HTML.xsl. אנו יכולים לשלב את הלוגיקה של XML2TOC.xsl עם Toc2HTML.xsl כדי להפיק את xml2toc2html.xsl !

הבה נוסיף עוד פונקציה (וכפתור נוסף) ל-ControlR2.asp :

```
case "Make HTML TOC From XML"
        Response.Write MakeTOCFromXML2HTML()
```

היישום של פונקציה מוצג בתדפיס 7.8.

תדפיס 7.8

```
199: Function MakeTOCFromXML2HTML()
200:         'convert XML file to TOC Listing file
201:         dim oXSL, fn
202:
203:         set oXSL = Server.CreateObject("FromScratch.XSLTransform")
204:
205:         'convert xhtml to xml via a stylesheet
206:         oXSL.InputFile = dataPath & ".xml"
207:         oXSL.XSLFile = fso.BuildPath(codeDir, "XML2TOC2HTML.xsl")
208:
209:         oXSL.Transform
210:
211:         fn = dataPath & ".toc.htm"
212:         oXSL.SaveOutputAsFile fn
213:
214:         MakeTOCFromXML2HTML = "TOC constructed as HTML"
215: End Function
```

פונקציה זו כמעט זהה לקודמת, אלא שהפעם בשורה 207 אנו משתמשים בגיליון סגנונות XSL שונה: xml2toc2html.xsl, כמוצג בתדפיס 7.9.

תדפיס 7.9

```
<?xml version="1.0"?>

<!--
  Given a chapter's .xml file, (e.g., chap3.xml) create a
  dynamic TOC in html
-->

<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/TR/WD-xsl"
  xmlns="http://www/w3/org/TR/REC-html40"
  result-ns="">

  <xsl:template match="/">
        <xsl:apply-templates />
  </xsl:template>

  <!-- catch any unknown tags -->
  <xsl:template match="*">
        <unknown><xsl:attribute name="tag"><xsl:node-name/></xsl:attribute>
              <xsl:apply-templates />
        </unknown>
```

```
</xsl:template>

<!-- our top level tag -->
<xsl:template match="book">
    <html>
        <head>
            <style>
                * {font-family:Verdana; font-size:10pt;}
                .listing {margin-bottom:3pt; cursor:hand;}
                .expand {color:red;}
            </style>
            <script><xsl:comment><![CDATA[
                function Expand()
                {
                  var e = window.event.srcElement.parentElement.
                  ➥parentElement;     // get to the "owning" div
                  var sign = e.children(0).children(0);     // the
                  ➥plus/minus sign span

        // if we were expanded, then hide the contents
        // don't do anything if there's no sign (ie. leaf node)
                switch (sign.innerText)
                {
                case '-':
                    sign.innerText = '+';
                    for (var i = 1; i < e.children.length; i++)
                    e.children(i).style.display="none";
                     break;
                case '+':
                    sign.innerText = '-';
                    for (var i = 1; i < e.children.length; i++)
                    e.children(i).style.display="";
                    break;
                }
            }
            function Highlight(on)
            {
                // change the background as the mouse passes
                ➥over selections
                event.srcElement.style.backgroundColor =
                ➥(on ? "yellow" : "");
                return false;
            }
            function ShowStory()
```

```
                {
                        // cause the rh frame to display the selected story -
                        ➥we have the story id as the listing element's id
                        var srcId = event.srcElement.id;


                        // tack on options to control the "mode" of display,
                        ➥based on checkboxes in BookControls.htm
                        // var NonCSS = parent.frames.bottom.
                        ➥NonCSS.checked;
                        // var CodeBlocks = parent.frames.bottom.
                        ➥CodeBlocks.checked;
                        // parent.frames.rh.location.href = "ShowStory.
                        ➥asp?StoryId=" + srcId + (NonCSS ? "&NonCSS=
                        ➥y" : "") + (CodeBlocks ? "&CodeBlocks=y" : "");

                        parent.location.href = "ShowStory.asp?StoryId=
                        ➥" + srcId;
                        return false;
                }
            ]]></xsl:comment></script>
        </head>
        <body>
                <xsl:apply-templates />
        </body>
    </html>
</xsl:template>

<!-- tags we can ignore -->
<xsl:template match="head|title|style|body|script|a|meta|link|div|b|i|spacerun|
➥tab|note|u|noteline|codeline|code|char"><xsl:apply-templates/></xsl:template>

<xsl:script>
    // global for maintaining the current indent level
    var indent = 0;
</xsl:script>


<!-- find the sections, create listings. Then create titles. -->
<xsl:template match="section">

    <div>
        <!-- This inner div controls the indenting level -->
        <div class="listing">
```

```
                    <xsl:attribute name="style">margin-left:<xsl:eval>
                    ➥indent</xsl:eval>em</xsl:attribute>
              <!-- The region that contains the +- sign for expanding -->
              <span class="expand" onclick="Expand()">
                    <xsl:choose>
                          <!--
                                If I have any listing nodes as children, then
                                ➥I am not a leaf, and so output a minus
sign
                                for the expand/contract control. If I am
                                ➥a leaf, then just use a small dot
                          -->
                          <xsl:when test="section">-</xsl:when>
                          <xsl:otherwise>&#8211;</xsl:otherwise>
                    </xsl:choose></span>
              <!--
                    This span holds the story title, and has mouse events
                    ➥to color it as the mouse moves over it
                    Clicking in this span will display the story in the rh pane,
                    ➥which is why we need the id attrib
              -->
              <span onmouseover="Highlight(true)" onmouseout=
              ➥"Highlight(false)" onclick="ShowStory()">
                    <xsl:attribute name="id"><xsl:value-of select=
                    ➥"@id"/></xsl:attribute>
                    <xsl:value-of select="./title"/></span>
              </div>
              <!--
                    bump up the indent before we recurse, then restore it
                    ➥when we're done
                    we have to use a global since there's no good way to pass
                    ➥in an argument
              -->
              <xsl:eval>indent += 2;"";</xsl:eval>
                    <!-- all we care about is section elements - we've already
                    ➥handled the title element above -->
                    <xsl:apply-templates select="section" />
              <xsl:eval>indent -= 2;"";</xsl:eval>
            </div>
        </xsl:template>
</xsl:stylesheet>
```

למרות שזהו תדפיס ארוך, יש הפתעות מעטות בלבד. המטרה שלנו היא לבצע את העבודה שקודם לכן בוצעה בשני קבצי XSL נפרדים. קובץ חדש זה מייצג חיבור פשוט של הלוגיקה שלהם.

10 השורות הראשונות מגדירות את קובץ ה-XSL וה-namespaces. נשים לב כי ה-namespace של הפלט הוא HTML, כפי שציפינו; התוצר הסופי שלנו מקובץ ה-XSL הוא קובץ HTML שניתן להצגה ע"ג הדפדפן.

הלוגיקה בשורות 12 עד 21 זהה לזו שבקבצים הקודמים, ולכן לא נרחיב בנקודה זו. בשורה 24 מבוצעת התאמה ל-book, כאשר אנו יכולים להיות בטוחים שוב, שמסמך הקלט יהיה פרק, כגון chap1.xml.

בעת ביצוע ההתאמה ל-book, אנו לא ניצור הפעם רכיב toc, אלא נוציא פלט למסמך ה-HTML DOM שלנו עם ה-HTML הנחוץ, שראינו קודם לכן ב-Toc2HTML.xsl. כך, אנו רואים את אותו הסגנון והתסריט שראינו בניתוח הקודם. הוא מוצג בשורות 23 עד 80.

בשורה 83 אנו מבצעים התאמה, ומתעלמים בדיוק מאותם הרכיבים שהתאימו, והתעלמנו מהם ב-xml2toc.xsl.

בשורה 92 מתבצעת התאמה לקטע section, וכאן הלוגיקה הופכת לצירוף של שתי הגישות הקודמות. מתבצעת התאמה לרכיב הקטע, מכיון שזה מה שיש לנו בקובץ ה-XML, והפלט יהיה של רכיבי ה-div הפנימי והחיצוני שאנו דורשים עבור ה-TOC הדינמית. לוגיקה זו נותרת ללא שינוי עד שורה 125, ובשורה 126 אנו סוגרים את גיליון סגנונות ה-XSL שלנו.

כפי שניתן לראות, אין שום דבר חדש בשימוש שלנו ב-XSL, אך בכל זאת יש חדש בכך שאנו מקבלים את הקלט שלנו ישירות מקובץ ה-XML של הפרק, והפלט שלנו הוא קובץ HTML שניתן להצגה בדפדפן.

הצעדים הבאים

ראינו כי XSL בשילוב עם שינויים של מסמך ה-DOM, מאפשר לנו גמישות רבה ביכולתנו לשנות XML בכמה דרכים. כעת משכל החלקים הבודדים קיימים ופועלים, זה הזמן להרכיב את היישום שלנו, וזה מה שנעשה בפרק הבא.

פרק 8
בניית היישום והרחבתו

בפרק זה:

- היישום
- בחינת היישום בפירוט
- יישום CodeBlocks
- הצעדים הבאים

עכשיו כמעט כל החלקים שבידינו עובדים ומוכנים עבור היישום.

זה הזמן לחבר אותם יחד ולהוסיף את יתר החלקים החסרים ליישום המלא שלנו.

זהו למעשה הפרק האחרון, שבו נגיע סוף סוף למטרתנו המיועדת - יישום לדפדוף בפרקי ספר, על גבי האינטרנט.

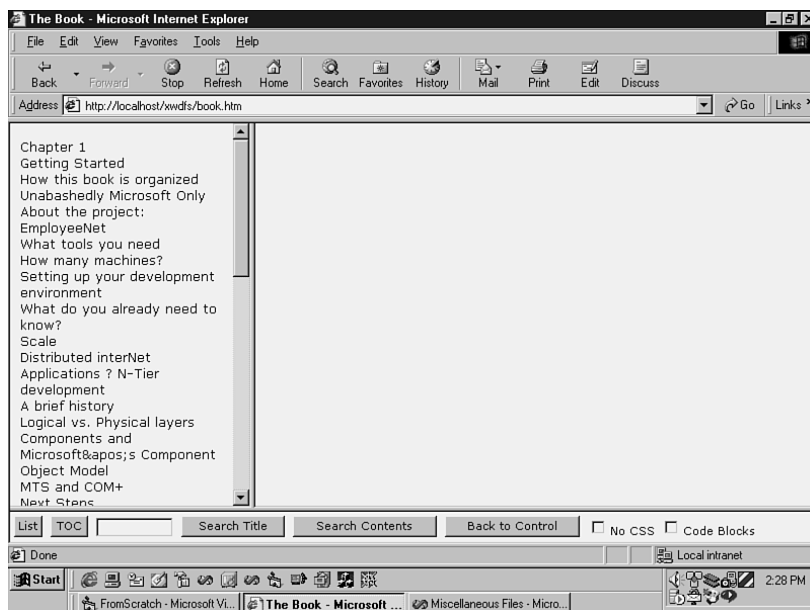
היישום

ניזכר רגע אחורה, בפרק 1, בו היישום BiblioTech, הוא מנוע חיפוש פשוט מבוסס-דפדפן, המאחזר סיפורים או תת-פרקים מתוך ספרים.

ראינו בכך שימוש עבור הגולש למציאת הסברים הולמים של מושגים, אשר יוכל לדפדף בין חלקי הספר, לקרוא, לגזור ולהעתיק טקסט.

BiblioTech תספק רשימה של כל הנושאים בספר, בצורת רשימה או בטבלת תוכן עניינים מתקפלת. כל נושא יהיה פעיל; לחיצה על הנושא תציג את המאמר הקשור. כמו כן, המשתמש יכול לחפש אחר מילים, או בכותרת, או בסופו של דבר, בכל מקום בטקסט.

תרשים 8.1 מציג כיצד נראה היישום כשהוא מופעל לראשונה.



תרשים 8.1: BiblioTech.



דרישות הדפסה של Que מכתיבות רזולוציה של 800x600, ולכן קיצצנו את הכפתור "List All", וקיצרנו את שדה הטקסט לתרשים זה.

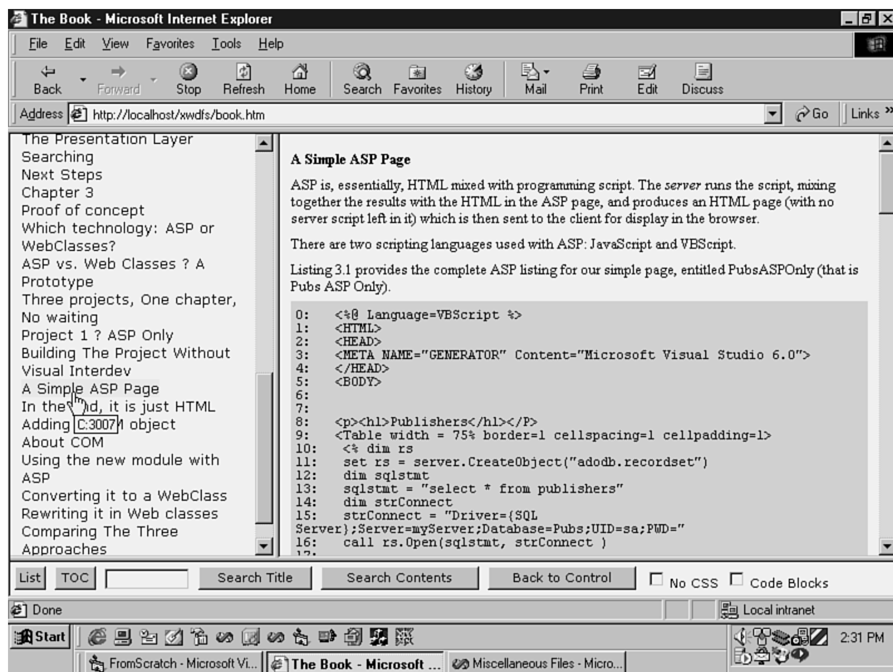
נזכיר כי מטרת היישום היא לסייע בידי המחבר במתן תשובות לשאלות הקוראים. אנו רוצים להיות מסוגלים לחפש אחר סיפורים על בסיס של מילים בטקסט או בכותרת, ואנו רוצים להבטיח שהמשתמש יוכל להעתיק קטעי קוד לזיכרון, ומאוחר יותר להדביק אותם למסמכים אחרים, או הודעות דואר אלקטרוני. כמו כן, נרצה להיות מסוגלים להשתמש ביישום זה בדפדפנים מיושנים, במקרה שנצטרך להריץ את היישום ב-IE3.

רישום סיפורים

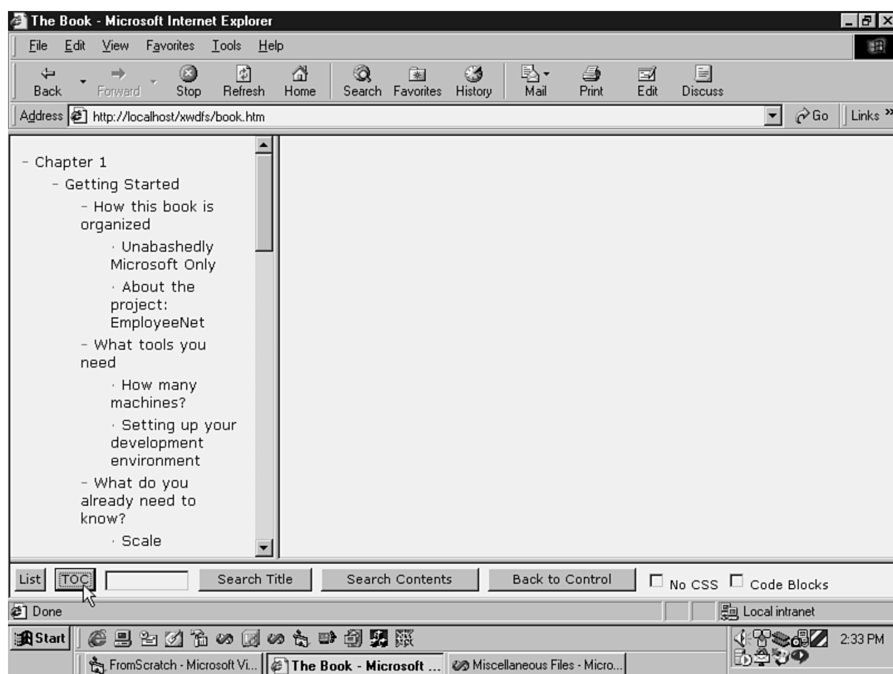
ניתן לגלול לאורך רשימת הנושאים, להאיר נושא על ידי השתהות מעליו עם סמן העכבר, וללחוץ עליו להצגת הסיפור, כמוצג בתרשים 8.2.

אפשר גם לראות את רשימת הסיפורים בטבלת תוכן עניינים מתקפלת, כמוצג בתרשים 8.3, על ידי לחיצה על הכפתור TOC.

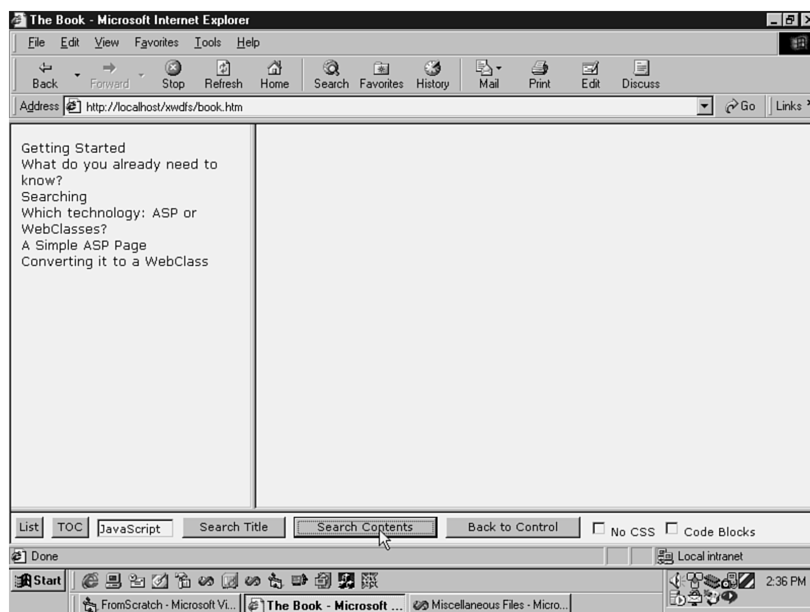
מאפיין חדש הוא היכולת לחפש אחר מאמר על פי הכותרת או התוכן שלו, על ידי מילוי תיבת הטקסט ולחיצה על הכפתור המתאים, כמוצג בתרשים 8.4.



תרשים 8.2: הצגת סיפור.



תרשים 8.3: טבלת תוכן העניינים.



תרשים 8.4: חיפוש.

כאן חיפשנו בתוכנם של כל הסיפורים אחר המילה "JavaScript", והיישום נתן רשימה של כל הסיפורים המכילים את המילה במקום כלשהו בטקסט.

דפדפנים מיושנים והעתקת קוד

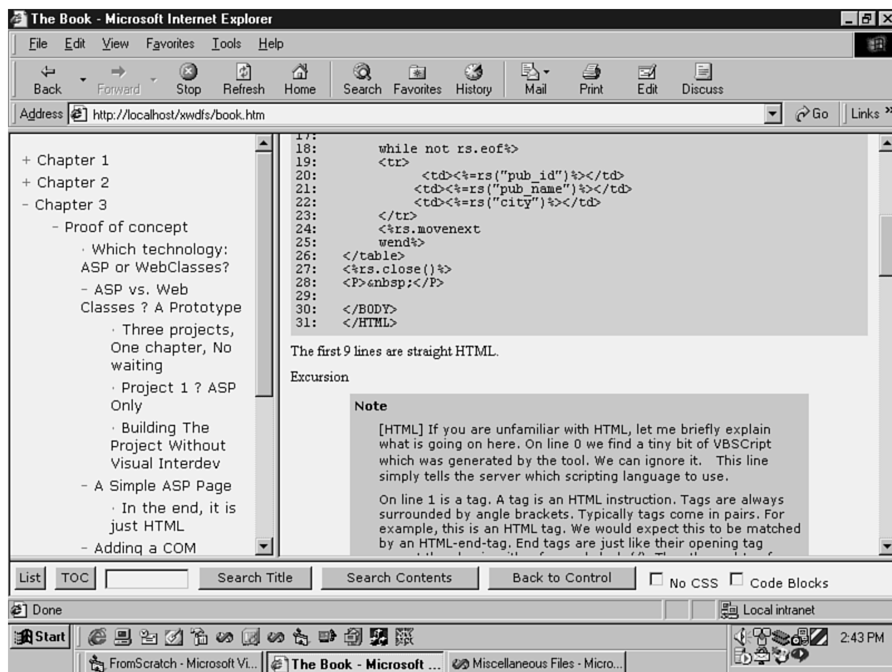
כדי לתמוך בדפדפנים שלא תומכים בגיליונות סגנון מדורגים, הוספנו תיבת סימון: "No CSS". כדי להבין את ההשפעה של תיבת הסימון, הבה נראה כיצד משתמשים בגיליונות סגנונות מדורגים כשהתיבה **לא** מסומנת.

תרשים 8.5 מראה קטע מהסיפור "A Simple ASP Page", המוצג באמצעות גיליונות סגנון מדורגים.

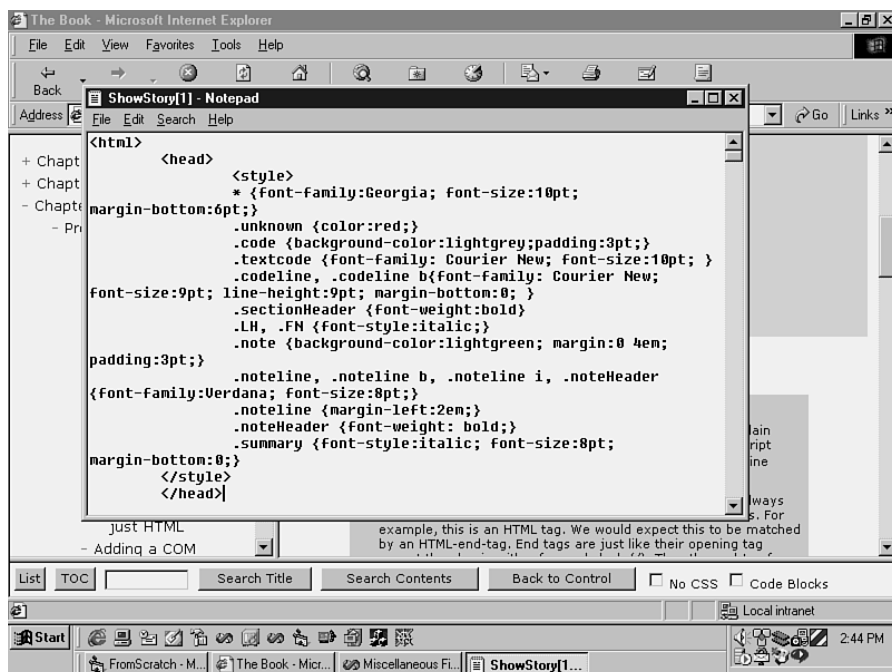
בפועל, האזורים המייצגים קוד מקור מוצג על גבי רקע אפור, וההערות מוצגות על גבי רקע ירוק. על ידי השימוש בגיליונות סגנון מדורגים, כפי שניתן לראות בתרשים 8.6, המראה את המקור המוצג במסגרת הימנית.

סימון "No CSS" מבטל את ה-CSS, ומנצל את פונקציונליות התצוגה של דפדפנים מיושנים, כפי שניתן לראות בתרשים 8.7.

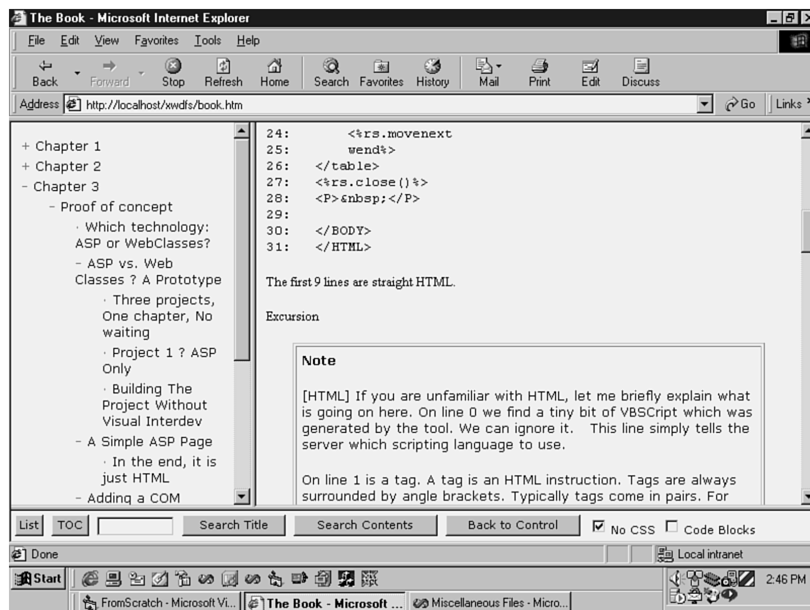
כשאנו מציגים קוד מקור, שום עיצוב CSS לא מסופק, אך כדי ליישר את הפלט, כך שיעוצב באופן מסודר יותר, אנו משתמשים בטבלאות, ובגופנים כדי לטפל בהצגת הטקסט.



תרשים 8.5: תצוגה עם גיליונות סגנון מדורגים (CSS).

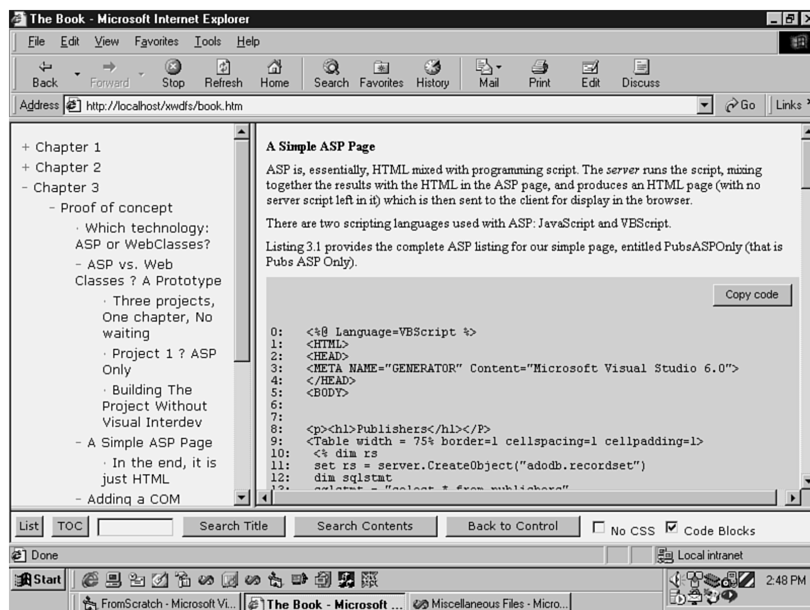


תרשים 8.6: הצגת קוד מקור.



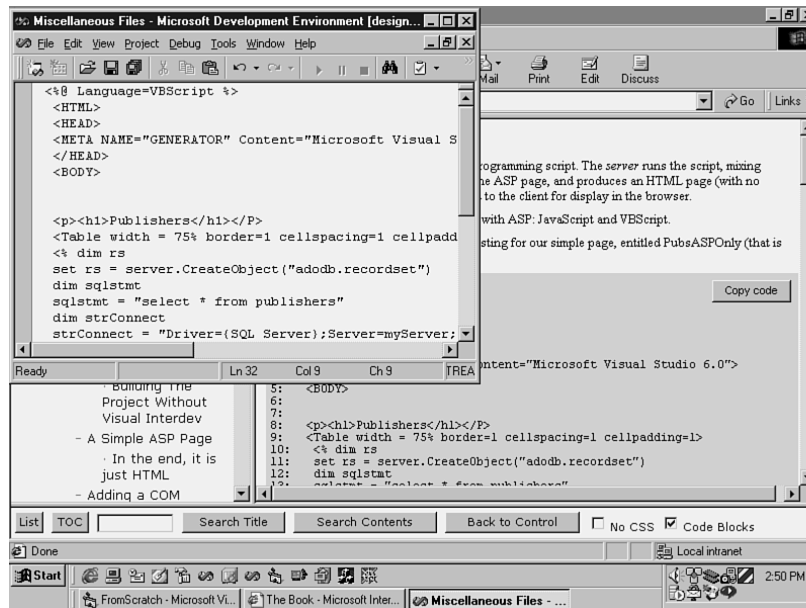
תרשים 8.7: ללא CSS.

נוסף לתיבת הסימון לביטול השימוש בעיצוב CSS, נוספה תיבת הסימון "Code Blocks". הדבר מוסיף את היכולת להעתיק קוד לתצוגת הלוח (Clipboard) ישירות מהתצוגה. סימון "Code Blocks" גורם לתצוגה לכלול כפתור בכל בלוק קוד, כפי שניתן לראות בתרשים 8.8.



תרשים 8.8: עם Code Blocks.

לחיצה על לחצן "Copy Code" בתדפיס הקוד מעתיקה את הקוד לזיכרון המחשב (ה-ClipBoard של Windows) **ללא מספרי השורות** וללא סימון HTML. זאת כדי להעתיק קוד נקי לזיכרון. כך, המשתמש יוכל, לנוחיותו, למשל, להדביק את הקוד בעורך קוד מקור, כמודגם בתרשים 8.9.



תרשים 8.9: הדבקת קוד המקור.

בחינת היישום בפירוט

ראינו חלק ניכר מהפונקציונליות של יישום זה בפרקים קודמים. חיבור של הכל יחד ליישום מבוסס מסגרות, הוספת החיפוש, הפונקציונליות של ביטול השימוש ב-CSS, ובלוקי הקוד הם כולם חדשים. בהמשך הפרק נבחן זאת.

את היישום ניתן להריץ על ידי לחיצה על "Go To Book Application" ב-Control.asp או על ידי מעבר ישיר ל-book.htm. הקובץ book.htm, המנהל את היישום, הוא קובץ מסגרות פשוט למדי, כמוצג בתדפיס 8.1.

תדפיס 8.1

```
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Developer Studio">
<META HTTP-EQUIV="Content-Type" content="text/html; charset=iso-8859-1">
<TITLE>The Book</TITLE>
</HEAD>
<frameset rows="*,32">
  <frameset cols="30%,*">
```

```
        <frame name="lh" src="StoryList.asp"/>
        <frame name="rh"/>
    </frameset>
    <frame name="bottom" src="BookControls.htm"/>
</frameset>
</HTML>
```

כפי שניתן לראות, קבוצת המסגרות החיצונית מכילה בתוכה קבוצת מסגרות פנימית, אשר בעצמה מכילה שתי מסגרות. הראשונה היא lh (עבור left-hand), המאותחלת ל-StoryList.asp. המסגרת השנייה היא rh (עבור right-hand). בקבוצת המסגרות החיצונית נמצאת גם המסגרת התחתונה הקרויה (באופן הולם) bottom, המאותחלת ל-BookControls.htm. למסגרת הימנית, rh, אין בתחילה אף דף, ולכן היא ריקה.

ביישום זה ישנם מספר דברים שצריך לבחון, אך מרכז הבקרה הוא BookControls.asp, ולכן נתחיל שם, כמוצג בתדפיס 8.2.

תדפיס 8.2

```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" content="text/html; charset=iso-8859-1">
<style>
* {font-family:Verdana; font-size:8pt; margin:2;}
</style>
</HEAD>
<script>
</script>
<BODY>
<input type="button" value="List All"
➥onclick="parent.frames.lh.document.location.href='StoryList.asp'">
<input type="button" value="TOC"
➥onclick="parent.frames.lh.document.location.href='ShowTOC.asp'">
<input id="searchText" size=20>
<input type="button" value="Search Title"
 onclick="parent.frames.lh.document.location.href=
➥'StoryList.asp?searchTitle=' + searchText.value">
<input type="button" value="Search Contents"
 onclick="parent.frames.lh.document.location.href='StoryList.asp?searchContents=
➥' + searchText.value">
<input type="button" value="Back to Control"
➥onclick="parent.location.href='Control.asp'">
<input type="checkbox" id="NonCSS">No CSS
<input type="checkbox" id="CodeBlocks">Code Blocks
</BODY>
</HTML>
```

השורות הראשונות הן HTML סטנדרטי. בשורה 10 אנו יוצרים כפתור עבור "List All" (המוצג כ-List בתרשים 8.1 כדי לחסוך במקום), והאירוע onClick שלו נקבע להצגת StoryList.asp במסגרת השמאלית.

בשורה 11 אנו רואים כפתור המסומן ב-TOC, המציג את ShowTOC.asp במסגרת השמאלית. באמצעות שני הכפתורים האלו, ניתן לעבור הלוך וחזור בין רשימת נושאים פשוטה ובין טבלת תוכן עניינים מוזחת. את ShowTOC.asp כבר ראינו, אבל StoryList.asp הוא חדש.

את StoryList.asp נבחן בעוד רגע, אך נשים לב כי בשורה 12 ישנו שדה קלט. בשורה 13 אנו יוצרים כפתור שנקרא "Search Title", אשר קורא ל-StoryList גם כן, אך הפעם מעביר את המחרוזת =SearchTitle משורשרת עם הערך של הטקסט שנכתב בשדה הטקסט.

כמו כן, בשורה 15 ישנו כפתור הקרוי SearchContents, הזהה גם כן לשורה 14, אך הפעם הוא מעביר את המחרוזת =SearchContents עם שרשור התוכן של שדה הטקסט.

התוצאה היא ש-StoryList יכול להיקרא באחת משלוש דרכים: לבד, עם =SearchTitle, או עם =SearchContents.

StoryList.asp

לשורות האחרונות של תדפיס 8.2 נחזור מאוחר יותר בפרק זה, אך הבה נבחן תחילה כיצד StoryList עובד, כמוצג בתדפיס 8.3.

תדפיס 8.3

```
<!-- #include file ="include.asp" -->
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" content="text/html; charset=iso-8859-1">
<style>
* {font-family:Verdana; font-size:10pt;}
.story {cursor: hand;}
</style>
</HEAD>
<script>
function Highlight(on)
{
 // change the background as the mouse passes over selections
 event.srcElement.style.backgroundColor = (on ? "yellow" : "");
 return false;
}
function ShowStory()
{
```

```
// cause the rh frame to display the selected story - we have the story id
➥as the listing element's id
var srcId = event.srcElement.id;

// tack on options to control the "mode" of display, based on checkboxes
➥in BookControls.htm
var NonCSS = parent.frames.bottom.NonCSS.checked;
var CodeBlocks = parent.frames.bottom.CodeBlocks.checked;

parent.frames.rh.location.href = "ShowStory.asp?StoryId=" + srcId +
➥(NonCSS ? "&NonCSS=y" : "") + (CodeBlocks ? "&CodeBlocks=y" : "");
return false;
}
</script>
<BODY>
<%
'construct a SQL WHERE clause based on the search arguments
dim where

'default to all stories
where = ""

'search by title
if Request("searchTitle") <> "" then
        where = "where title like '%" & Replace(Request("searchTitle"), "'", "''") & "%'"
end if

'search by text within the bodies
if Request("searchContents") <> "" then
        where = "where TaglessText like '%" &
        ➥Replace(Request("searchContents"), "'", "''") & "%'"
end if

'get the selected stories and list their titles. make the id of each element be
➥the story id
set rs = DBConn.Execute("select StoryId, SectionLevel, Title from Stories
➥" & where & " order by StoryId")
do until rs.eof
%>
<div><span class="story" id="<% =rs("StoryId") %>"
➥title="<% =rs("SectionLevel") & ":" & rs("StoryId") %>"
 onmouseover="Highlight(true)" onmouseout="Highlight(false)"
➥onclick="ShowStory()"><% =rs("Title") %></span></div>
<%
```

```
54:          rs.MoveNext
55:   loop
56:  %>
57:  </BODY>
58:  </HTML>
```

הקובץ מחולק לשלושה חלקים: תסריט בצד הלקוח, תסריט בצד השרת, ו-HTML. השניים האחרונים חופפים מעט, שכן התסריט בצד הלקוח וה-HTML מעורבים בקבצי ASP.

אנו מתחילים בשורות 4-7 עם קטע עיצוב CSS קצר בו אנו קובעים את סגנון ברירת המחדל (על ידי שימוש ב- *) ואת הסגנון עבור סיפורים (הקובע את סמן העכבר לסמן של יד כשהעכבר מעל הטקסט, mouseOver).

שורות 9-28 מציגות את חלק התסריט שבצד הלקוח, המורכב משתי שיטות: Highlight ו-ShowStory. הן זהות ביסודם כפי שראינו בפרק הקודם.

Highlight() משמשת לתיחום של כותרת הסיפור בעת mouseOver; ShowStory משמשת לבחירת הסיפור כשהכותרת נלחצת. כל אחת מהן נקבעת כהליך לטיפול באירוע בשורות 51-52. נחזור לשורות אלו בעוד זמן קצר.

ShowStory מתחילה בשורה 19 בקביעת ה-ID של הרכיב שעורר את ההליך: כלומר, הכותרת שנבחרה. בשורה 22 אנו יכולים לראות האם תיבת הסימון NonCSS סומנה, ובשורה 23 אנו יכולים לראות את תיבת הסימון CodeBlocks.

העבודה נעשית בשורה 25. אנו קובעים את המסגרת הימנית להצגת הדף ShowStory.asp, תוך העברה לדף את הפרמטר StoryID שנקבע ל-id שהישגנו בשורה 19, מלווה באחד או יותר משני דגלים אפשריים. אם NonCSS מסומנת, אז לפרמטר המחרוזת משורשר "&NonCSS=y", ואם CodeBlocks מסומנת, אז משורשר "&CodeBlock=y".



האופרטור המשולש (מיוצג על ידי סימן שאלה - ?) הוא מושג ב-JavaScript, Java ו- ++C. הוא מורכב משלושה ביטויים: המבחן, הערך המוחזר במקרה של true, וערך המוחזר במקרה של false. במקרה המוצג לפנינו, למשל, אנו בודקים את הערך של NonCSS; אם הוא true אנו מחזירים את הערך "NonCSS=y"; אם הוא false אנו מחזירים את המחרוזת הריקה.

כשדף ASP נקרא עם מחרוזת כמו "ShowStory.asp?StoryID=5&NonCSS=y" אוסף אובייקטי בקשות ה-ASP, QueryString, יכיל שתי יישויות: StoryID ו-NonCSS. הערך של הקודם יהיה 5, והערך של השני יהיה y.

בשורה 30 אנו מתחילים את התסריט בצד השרת אשר בונה הצהרת SQL להפעלה בשרת, בהתאם לערך של המשתנים SearchTitle ו-SearchContents באובייקט בקשת ה-ASP. ערכים אלו נקבעים כש-StoryList נקראת על ידי bookControls.htm כמוצג בשורה 14 בתדפיס 8.2:

```
14:  onclick="parent.frames.lh.document.location.href=
     ➥'StoryList.asp?searchTitle=' + searchText.value">
```

הצהרת ה-SQL עצמה נקראת בשורה 48, ומחזירה רשומות עם שלושה שדות כל אחת: StoryID, SectionLevel, ו-Title. בשורות 49-55 אנו עוברים באיטרציה על הרשומות האלו, ויוצרים <div> עבור כל רשומה, שיחזיק את הכותרות של סיפור. זהו המקום בו אנו קובעים את הליכי הטיפול באירועים.

יישום XSL עבור דפדפנים מיושנים

הבה נבצע חזרה קצרה: book.htm ו-bookControl.asp עובדים עם ShowTOC או StoryList ליצירת רשימה של כותרות במסגרת הימנית.

כשהמשתמש לוחץ על סיפור מסוים, ShowStory נקראת, והמצב של שתי תיבות הסימון מועבר כפרמטר. בפרק הקודם, כשבחנו את ShowStory, התעלמנו מהפרמטרים האלו, והנחנו שהם ריקים (למעשה, זה מייצג מצב בו שתי תיבות הסימון אינן מסומנות).

הבה נבחן מה קורה כאשר תיבת הסימון NonCSS **כן** מסומנת, וזהו המקרה בו אנו רוצים שהסיפור יוצג, ללא שימוש בגיליונות סגנונות מדורגים. מכיון ש-ShowStory היא קצרה, להלן עותק מלא שלה בתדפיס 8.4.

תדפיס 8.4

```
<!-- #include file ="include.asp" -->
<%
  'ShowStory.asp - retrieves the XML for single story from the database,
  'converts it into HTML via an XSL stylesheet, and displays the results

  dim storyId, xml, oXSL, styleSheetName

  storyId = Request("storyId")

  'get the XML from the database
  set rs = DBConn.Execute("select xml from stories where storyId = " & storyId)
  xml = rs("xml").value
  rs.close

  'transform to HTML via the XSL stylesheet
  set oXSL = Server.CreateObject("FromScratch.XSLTransform")
  oXSL.InputXML = xml
  styleSheetName = "Story2HTML.xsl"
  if Request("NonCSS") <> "" then styleSheetName = "Story2HTMLNonCSS.xsl"
  oXSL.XSLFile = fso.BuildPath(codeDir, styleSheetName)
  oXSL.Transform

  'display the HTML
  response.write oXSL.OutputXML
%>
```

נזכיר כי אנו משיגים את הסיפור ממסד הנתונים בשורות 7-12, ואחר כך, בשורות 15-16, אנו קוראים אותו ממסד הנתונים לתוך מסמך XML DOM.

בשורה 17 אנו קובעים את גיליון הסגנונות ל-Story2HTML.xsl כפי שראינו קודם לכן. בשורה 18 אנו מסמנים את משתנה הבקשה NonCSS.

```
18:   if Request("NonCSS") <> "" then styleSheetName = "Story2HTMLNonCSS.xsl"
```

בפעם האחרונה הנחנו שהוא היה ריק, לכן לא היתה לו כל השפעה. הפעם נניח שתיבת הסימון מסומנת, ולמשתנה הזה יש עכשיו את הערך "y"; במקרה הזה המשתנה styleSheetName ישונה ל-Story2HTMLNonCSS.xsl, הקורא לגיליון סגנונות XSL שונה, כפי שניתן לראות בתדפיס 8.5.

תדפיס 8.5

```
0:  <?xml version="1.0"?>
1:  <!-- Stylesheet for displaying XML stories, but for a down-level browser - no use
    ➥of styles -->
2:  <xsl:stylesheet
3:   xmlns:xsl="http://www.w3.org/TR/WD-xsl"
4:   xmlns="http://www/w3/org/TR/REC-html40"
5:   result-ns="">
6:
7:   <!-- default rule for text nodes - just output the text -->
8:   <xsl:template match="text()">
9:          <xsl:value-of/>
10:  </xsl:template>
11:
12:  <!-- the root node - output the structure of the HTML page and begin
     ➥the recursion -->
13:  <xsl:template match="/">
14:  <html>
15:         <xsl:apply-templates/>
16:  </html>
17:  </xsl:template>
18:
19:  <!-- sentinel for any left over tags -->
20:  <xsl:template match="*">
21:         <div><font color="red">
22:                !!<xsl:node-name/>!!
23:                <xsl:apply-templates />
24:         </font></div>
25:  </xsl:template>
26:
27:  <!--
28:         The top level element - might be story or book depending on who's calling
29:         Just wraps the content inside <body> tags
```

```
        Then finishes with some summary info about numbers of nodes
        ➥of different kinds
 -->
<xsl:template match="story|book">
<body>
        <font face="Georgia" size="2">
        <xsl:apply-templates/>
        <p><i>
                Section summary
                <br/><xsl:eval>this.selectNodes("//div").length</xsl:eval> paragraphs
                <br/><xsl:eval>this.selectNodes("//code").length
                ➥</xsl:eval> code blocks
                <br/><xsl:eval>this.selectNodes("//note").length</xsl:eval> notes
        </i></p></font>
</body>
</xsl:template>

<!-- We don't need to do anything special at the section level -->
<xsl:template match="section">
        <xsl:apply-templates/>
</xsl:template>

<!--
        Our basic text paragraph element. We maintain the class attribute,
        ➥which is really the same
        as the original Word style name
 -->
<xsl:template match="div">
        <p>
                <xsl:apply-templates/>
        </p>
</xsl:template>

<xsl:template match="div[@class='LH'] | div[@class='FN']">
        <p>
                <i><xsl:apply-templates/></i>
        </p>
</xsl:template>

<!-- we ignore PD - these are publishing directions, e.g. "begin note" -->
<xsl:template match="div[@class='PD']">
</xsl:template>

<!--
```

```
        These are our specialized content types for code blocks and notes.
        Each now needs its own font commands
-->
<xsl:template match="codeline">
        <div><xsl:apply-templates/></div>
</xsl:template>

<xsl:template match="noteline">
        <p><xsl:apply-templates/></p>
</xsl:template>

<xsl:template match="code">
        <p><font face="Courier New"><xsl:apply-templates/></font></p>
</xsl:template>

<!-- The note element is a little special, because we want to put in a header
➥that says "Note" -->
<xsl:template match="note">
        <center><table border="1" cellpadding="5"
        ➥width="90%"><tr><td><font face="Verdana" size="2">
        <b>Note</b>
        <xsl:apply-templates/>
        </font></td></tr></table></center>
</xsl:template>

<!-- Titles for sections are bolded -->
<xsl:template match="title">
        <div><font size="3"><b><xsl:apply-templates/></b></font></div>
</xsl:template>

<!-- underline elements are used to denote code in the text - turn these into
➥their own special spans -->
        <xsl:template match="u">
                <span><font face="Courier New">
                <xsl:apply-templates/></font></span>
        </xsl:template>

<!-- handle vanilla HTML-like tags - just map them to the same thing -->
<xsl:template match="b|i|sub|sup|br">
        <xsl:element>
                <xsl:apply-templates/>
        </xsl:element>
</xsl:template>

```

```
<!-- spaceruns and tabs get mapped into sequences of  s.
➥We approximate tabs with 4 spaces -->
<xsl:template match="spacerun">
	<span><xsl:eval>Repeat(" ",
	➥this.getAttribute("len"))</xsl:eval></span>
</xsl:template>

<xsl:template match="tab">
	<span><xsl:eval>Repeat(" ", 4)</xsl:eval></span>
</xsl:template>

<!-- translate special chars into entities for display in HTML - we use
➥numbered entities so we don't have to
	explicitly define them -->
<xsl:template match="char">
	<xsl:choose>
		<xsl:when test=".[@type = 'emDash']">&#8212;</xsl:when>
		<xsl:when test=".[@type = 'bullet']">&#8226;</xsl:when>
		<xsl:when test=".[@type = 'ellipsis']">&#8230;</xsl:when>
		<!-- we'll just use 2 nbsps for emSpace - #8195 doesn't seem
		➥to work as documented -->
		<xsl:when test=".[@type = 'emSpace']">  </xsl:when>
		<xsl:when test=".[@type = 'smartApos']">&#146;</xsl:when>
		<xsl:when test=".[@type = 'smartLQuote']">&#147;</xsl:when>
		<xsl:when test=".[@type = 'smartRQuote']">&#148;</xsl:when>
		<xsl:otherwise><span class="unknown">char: <xsl:value-of
		➥select="@type"/></span></xsl:otherwise>
	</xsl:choose>
</xsl:template>

<xsl:script>
<![CDATA[
// returns a string consisting of n copies of t
function Repeat(t, n)
{
	var r = "";
	while (n-- > 0)
		r += t;
	return r;
}
]]>
</xsl:script>

</xsl:stylesheet>
```

גיליון הסגנונות הובא כאן במלואו, למרות שרק מעט ממנו שונה מ-StoryToHTML שהובא בפרק הקודם. זה יספק הקשר לשינויים המודגשים בתדפיס.

השינוי הראשון והברור ביותר הוא שהתגית <html> המוצגת בשורות 14 ו-16 מכילה עכשיו רק את הערכים המוחזרים מהקריאה ל-apply-templates. בגירסה הקודמת מצאנו קבוצה נרחבת של רכיבי סגנון.

באופן לא מפתיע, אנו חייבים לקבוע את הגופן באופן מפורש, כמו לדוגמה בשורה 34. אחר כך נגלה שבעוד שהיינו מסוגלים להשתמש ברכיבי <div> עם תכונת המחלקה בגירסה הקודמת, כאן נשתמש ברכיבי <p>, ובעיצוב מפורש, כמו השימוש ברכיב <I> המוצג בשורה 36. מאחר ואין לנו את היכולת לשנות את העיצוב בהתבסס על המחלקה של הרכיב, אנו חייבים לחפש אחר רכיבים ספציפיים בתוך מסמך המקור ולקבוע את העיצוב בהתאם. לכן, בשורות 60-64 אנו קובעים את הרכיבים LH ו-FN לתצוגה בהטיה.

בשורות 87-92 אנו ממקמים את ההערות בתוך קופסה על ידי יצירת טבלה עם גבולות. קוד inline מצוין על ידי שינוי הגופן ל-Courier New כפי שניתן לראות בשורות 100-102.

בקצרה, בעוד שעלינו להיות מעט מתוחכמים בנוגע לדרך שינוי ה-HTML בדפדפנים מיושנים, אין כאן הפתעות גדולות. ניתן לדמיין בקלות יצירה של גיליונות סגנון XSL דומים למעבר לכל מפרט של הפלט.

יישום CodeBlocks

גיליון ה-XSL ליישום CodeBlocks הוא מעט יותר מורכב. כדי להתחיל לכתוב גיליון סגנונות כזה, אנו חייבים לשקול מה אנחנו רוצים בדף ה-HTML שהוא מפיק.

המטרה שלנו היא ליצור כפתור שיעתיק את הקוד לזיכרון. הבה נניח שהמפרט דורש IE5. ניסיון לגרום להתאמה ל-IE4 ול-Netscape יהיה, אם לא בלתי אפשרי, לפחות לא נעים. ניתן כמובן, לכתוב אובייקט לקוח מותאם אישית, אבל זה הרבה יותר קל לתת ל-IE5 לעשות את העבודה.

יצירת CodeBlocks ב-HTML

נזכיר מהפרקים הקודמים שכשאנו מציגים קוד ב-HTML, אנו יוצרים רכיב <div> חיצוני עם התכונה class שערכה הוא code, ורכיב <div> פנימי לכל שורת קוד, כמוצג בקטע להלן מה-HTML המתקבל:

```
<div class="code"><div class="codeline">
```

לטיפול בכפתור, נוסיף לרכיב <div> החיצוני כפתור שערכו (ולכן הטקסט שלו) יהיה Copy Code. נשייך הליך לטיפול במאורעות עבור לחיצה על הכפתור שיקרא לפונקציית התסריט בצד הלקוח CopyCode, ונעביר מזהה ייחודי לבלוק הקוד. ה-HTML המתקבל נראה כך:

```
<div class="code"><table width="100%"><tr><td align="right">
➥<input type="button" value="Copy Code" class="button"
➥onClick="CopyCode('CodeBlock84542208')"/></td></tr></table>
```

נשים לב כי הכפתור מוקם בתוך טבלה כך שנוכל לגרום לו לזוז ימינה. לשיטה CopyCode מועבר מזהה ייחודי: המילה CodeBlock מלווה במספר מזהה ייחודי. בעוד רגע נראה מהיכן מגיע המזהה הזה.

כשהכפתור נלחץ, השיטה CopyCode תיקרא, כמוצג בתדפיס 8.6.

תדפיס 8.6

```
<html>
 <head>
     <style>
     * {font-family:Georgia; font-size:10pt; margin-bottom:6pt;}
     .unknown {color:red;}
     .code {background-color:lightgrey;padding:3pt;}
     .textcode {font-family: Courier New; font-size:10pt; }
     .codeline, .codeline b{font-family: Courier New; font-size:9pt;
     ➥line-height:9pt; margin-bottom:0; }
     .sectionHeader {font-weight:bold;}
     .LH, .FN {font-style:italic;}
     .note {background-color:lightgreen; margin:0 4em; padding:3pt;}
     .noteline, .noteline b, .noteline i, .noteHeader {font-family:Verdana;
     ➥font-size:8pt;}
     .noteline {margin-left:2em;}
     .noteHeader {font-weight: bold;}
     .summary {font-style:italic; font-size:8pt; margin-bottom:0;}
     .button {font-family: Arial; font-size:8pt;}
 </style>
     <script><!--
     function CopyCode(id)
     {
         // find the appropriate block of text
         var d = theCodeData.selectSingleNode("//codeblock[@id='" + id + "']");
         // extract the code by concatenating all the text - this also resolves
         ➥the HTML quoting
         var s = d.text;
         // stash the text in an element, so we can create a text range
         trElement.innerText = s;
```

```
            // the only way I know of to get something on the clipboard is
            ➥to exec a command on a text range
            var tr = trElement.createTextRange();
            tr.execCommand("Copy");
        }
-->></script>
</head>
```

תדפיס 8.6 הוא הקטע הראשי של קובץ ה-HTML. הוא כולל את אזור הסגנונות, כמו גם את התסריט שבצד הלקוח, במקרה זה השיטה CopyCode.

CopyCode מסתמכת על קיומו של אי (island) XML במסמך ה-HTML. כפי שצוין בפרק הראשון, אי XML הוא המצאה של Microsoft, הנתמכת על ידי IE5, כדי לשבץ אזור של XML בקובץ HTML. אי XML מוגדר על ידי תגית ה-HTML, <xml>. אנו רואים זאת מאוחר יותר בקובץ ה-HTML, כמוצג בתדפיס 8.7. כדי להבין כיצד זה עובד נצטרך לבחון את תדפיס 8.6 ותדפיס 8.7 ביחד.

תדפיס 8.7

```
        <div class="summary">
                Section summary
                <br/>34 paragraphs
                <br/>2 code blocks
                <br/>1 notes
        </div>
        <xml id="theCodeData"><codedata>
                        <codeblock id="CodeBlock84542208">
&lt;%@ Language=VBScript %&gt;
&lt;HTML&gt;
&lt;HEAD&gt;
&lt;META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0"&gt;
&lt;/HEAD&gt;
&lt;BODY&gt;


&lt;p&gt;&lt;h1&gt;Publishers&lt;/h1&gt;&lt;/P&gt;
&lt;Table width = 75% border=1 cellspacing=1 cellpadding=1&gt;
&lt;% dim rs
set rs = server.CreateObject("adodb.recordset")
dim sqlstmt
sqlstmt = "select * from publishers"
dim strConnect
strConnect = "Driver={SQL Server};
➥Server=myServer;Database=Pubs;UID=sa;PWD="
call rs.Open(sqlstmt, strConnect )
```

```
25:
26:     while not rs.eof%&gt;
27:     &lt;tr&gt;
28:         &lt;td&gt;&lt;%=rs("pub_id")%&gt;&lt;/td&gt;
29:         &lt;td&gt;&lt;%=rs("pub_name")%&gt;&lt;/td&gt;
30:         &lt;td&gt;&lt;%=rs("city")%&gt;&lt;/td&gt;
31:     &lt;/tr&gt;
32:     &lt;%rs.movenext
33:     wend%&gt;
34:  &lt;/table&gt;
35:  &lt;%rs.close()%&gt;
36:  &lt;P&gt;&amp;nbsp;&lt;/P&gt;
37:
38:  &lt;/BODY&gt;
39:  &lt;/HTML&gt;
40: </codeblock>
41:                         <codeblock id="CodeBlock84549536">
42:         &lt;td&gt;&lt;%=rs("pub_id")%&gt;&lt;/td&gt;
43: </codeblock>
44:                 </codedata></xml>
45:         <input type="hidden" id="trElement"/>
46:   </body>
47: </html>
```

בשורה 6 בתדפיס 8.7, ניתן לראות את אי ה-XML מוגדר באמצעות הרכיב xml:

```
6:          <xml id="theCodeData"><codedata>
```

theCodeData הוא ה-id או השם המזהה של אי ה-XML. בתוך ה-HTML אנו יכולים להתייחס לאי ה-XML באמצעות ה-id, theCodeData. בין התגית xml הפותחת בשורה 6, ובין התגית xml הסוגרת בשורה 44, אנו בתוך אי של XML, וכל הרכיבים חייבים להיות רכיבי XML, ולא HTML.

רכיב ה-XML הראשון שאנו רואים הוא גם כן בשורה 6; זהו רכיב ה-XML <codedata>. הרכיב codedata משמש לתיחום של אזור הקוד השמור שלנו, והוא מספק את הרכיב הבודד ברמה העליונה הנדרש על ידי XML, בדיוק כמו book ו-story בפרקים הקודמים. בתוך הרכיב codedata יהיו מוכלים אחד או יותר רכיבי codeblock. כל רכיב codeblock תוחם בלוק מסוים של קוד. כל codeblock מזוהה על ידי id ייחודי, כך שאנו יכולים להתייחס אליו בנפרד. בתוך בלוק הקוד מצוי הקוד ללא מספרי שורות וללא תגיות HTML. תוכן ה-codeblock הוא שיועתק לזיכרון המחשב.

אנו נניח לעת עתה לשאלה איך יצרנו את אי ה-XML עם הרכיבים codedata ו-codeblock המוכלים בו; נחזור לכך כשנסתכל על גיליון ה-XSL. כעת, נצא מנקודת ההנחה שאי ה-XML קיים. בחזרה לתדפיס 8.6 והיישום של CopyCode, בשורה 21 אנו רואים שאנו יכולים לקרוא ל-selectSingleNode על אי ה-XML.



נזכיר ש-theCodeData הוא רכיב מסוג XML, שהוא בעצמו רכיב **HTML**, ולכן יכול להיקרא בתוך תסריט DHTML.

SelectSingleNode אינה שיטה ב-HTML; זוהי שיטה ב-XML. כיצד זה ייתכן שאנו קוראים לשיטה ב-XML על רכיב HTML? מסתבר שמאפיין ברירת המחדל של רכיב ה-XML הוא מסמך XMLDocument. מאפיין זה מחזיר הפנייה למסמך ה-XML DOM הבסיסי.

כך, הקריאה theCodeData.SelectSingleNode זהה בדיוק לקריאה theCodeData.XMLDocument.selectSingleNode.

כך שאנו לא קוראים ל-SelectSingleNode על האי, אלא על XMLDocument, שהוא אובייקט XML.

בעודנו בררניים, אנו יכולים לקרוא ישירות ל-theCodeData, מכיון ש-DHTML מניחה שאובייקט ברירת המחדל הוא document.all. כך, ההצהרה התמימה theCodeData.SelectSingleNode היא למעשה קיצור ל-document.all.theCodeData.XMLDocument.SelectSingleNode.

אנו מעבירים הצהרת XQL (XML Query Language) ל-selectSingleNode, המחפשת אחר הרכיב codeblock הראשון מבין הצאצאים (לא רק הבנים הישירים) של XMLDocument, עם התכונה id שנקבעה ל-id שהעברנו כפרמטר של CopyCode. בקיצור, היא מוצאת את ה-codeblock עם ה-id המתאים.

בשורה 23 אנו מחלצים את הטקסט מה-codeblock המתאים. נשים לב כי בהערה בשורה 22 הקריאה למאפיין text מפרידה גם את ציטוט ה-HTML, כך למשל, < מוחזר כ- < בדיוק כמו שהיינו רוצים.

שורות 25-28 הן ספציפיות ל-IE5 והכרחיות להעתקת הטקסט לתצוגת הלוח. בשורה 25 אנו קובעים את הרכיב החבוי trElement למחרוזת על ידי שימוש במאפיין ה-DHTML, innerText. בשורה 27 אנו קוראים ל-createTextRange על רכיב זה, ומחזירים תחום טקסט עליו נקרא ל-execCommand, אשר מפעילה את הפקודה Copy ומעתיקה את הטקסט לזיכרון.

יצירת ה-HTML על ידי טרנספורמציית XSL

עכשיו כשאנו יודעים כיצד ה-HTML הסופי צריך להיראות, קל יותר ליצור את גיליון ה-XSL שלנו. שוב, אנו מתחילים עם Story2HTML.xsl ומשנים אותו ליצירת Story2HTMLCodeBlocks.xsl, כמוצג בתדפיס 8.8.

תדפיס 8.8

```
0: <?xml version="1.0"?>
1: <!-- Stylesheet for displaying XML stories augmented to keep a unadorned version
   ➥of code blocks as an XML island -->
2: <xsl:stylesheet
3:   xmlns:xsl="http://www.w3.org/TR/WD-xsl"
```

```
xmlns="http://www/w3/org/TR/REC-html40"
result-ns="">

<!-- default rule for text nodes - just output the text -->
<xsl:template match="text()">
        <xsl:value-of/>
</xsl:template>

<!-- the root node - output the structure of the HTML page and begin
➥the recursion -->
<xsl:template match="/">
<html>
<head>
<style>
        * {font-family:Georgia; font-size:10pt; margin-bottom:6pt;}
        .unknown {color:red;}
        .code {background-color:lightgrey;padding:3pt;}
        .textcode {font-family: Courier New; font-size:10pt; }
        .codeline, .codeline b{font-family: Courier New; font-size:9pt;
        ➥line-height:9pt; margin-bottom:0; }
        .sectionHeader {font-weight:bold}
        .LH, .FN {font-style:italic;}         <!-- listing header, figure caption -->
        .note {background-color:lightgreen; margin:0 4em; padding:3pt;}
        .noteline, .noteline b, .noteline i, .noteHeader {font-family:Verdana;
        ➥font-size:8pt;}
        .noteline {margin-left:2em;}
        .noteHeader {font-weight: bold;}
        .summary {font-style:italic; font-size:8pt; margin-bottom:0;}
        .button {font-family: Arial; font-size:8pt;}
</style>
<script><xsl:comment><![CDATA[
        function CopyCode(id)
        {
                // find the appropriate block of text
                var d = theCodeData.selectSingleNode
                ➥("//codeblock[@id='" + id + "']");
                // extract the code by concatenating all the text - this also
                ➥resolves the HTML quoting
                var s = d.text;
                // stash the text in an element, so we can create a text range
                trElement.innerText = s;
                // the only way I know of to get something on
                ➥the clipboard is to exec a command on a text range
                var tr = trElement.createTextRange();
```

```
                tr.execCommand("Copy");
        }
]]></xsl:comment></script>
</head>
        <xsl:apply-templates/>
</html>
</xsl:template>

<!-- sentinel for any left over tags -->
<xsl:template match="*">
        <div class="unknown">
                !!<xsl:node-name/>!!
                <xsl:apply-templates />
        </div>
</xsl:template>

<!--
        The top level element - might be story or book depending on who's calling
        Just wraps the content inside <body> tags
        Then follows with some summary info about numbers of nodes
        ➥of different kinds
        The last piece is a set of <XML> islands that stores a copy
        ➥of any code blocks to
        allow copying of unformatted code to the clipboard
 -->
<xsl:template match="story|book">
<body>
        <xsl:apply-templates/>
        <div class="summary">
                Section summary
                <br/><xsl:eval>this.selectNodes("//div").length</xsl:eval> paragraphs
                <br/><xsl:eval>this.selectNodes("//code").length
                ➥</xsl:eval> code blocks
                <br/><xsl:eval>this.selectNodes("//note").length</xsl:eval> notes
        </div>
        <!-- create an XML island that contains the code for
        ➥each code block that happens to be in this document -->
        <xml id="theCodeData"><codedata>
                <xsl:for-each select="//code">
                        <codeblock>
                                <!-- give each <codeblock> a unique id that
                                ➥can be referenced from the
                                ➥visible rendering -->
```

```
                    <xsl:attribute name="id">CodeBlock<xsl:eval>
                    ➥uniqueID(this)</xsl:eval></xsl:attribute>
                    <!-- for each codeline, we want to insert
                    ➥the text, but we have a lot of extra white
                    ➥space, so use some script -->
                    <xsl:apply-templates select="*">
                        <xsl:template match="codeline">
                            <xsl:eval>Expand(this)</xsl:eval>
                        </xsl:template>
                    </xsl:apply-templates>
                </codeblock>
            </xsl:for-each>
        </codedata></xml>
        <!-- and we need an element on which we can construct
        ➥a textRange, so we can copy to the clipboard -->
        <input type="hidden" id="trElement" />
</body>
</xsl:template>

<!--
        Sections just map into DIVs with their own class. We'll keep
        ➥the level attribute incase we ever
        want to do formatting based on it
 -->
<xsl:template match="section">
<div class="section"><xsl:attribute name="level">
➥<xsl:value-of select="@level"/></xsl:attribute>
        <xsl:apply-templates/></div>
</xsl:template>

<!--
        Our basic text paragraph element. We maintain the class attribute,
        ➥which is really the same
        as the original Word style name
 -->
<xsl:template match="div">
        <div>
                <xsl:attribute name="class">
                ➥<xsl:value-of select="@class"/></xsl:attribute>
                <xsl:apply-templates/>
        </div>
</xsl:template>

<!-- we ignore PD - these are publishing directions, e.g. "begin note" -->
```

```
<xsl:template match="div[@class='PD']">
</xsl:template>

<!--
        These are our specialized content types for code blocks and notes.
        We just change them into divs with corresponding styles
-->
<xsl:template match="codeline|noteline">
<div><xsl:attribute name="class"><xsl:node-name/></xsl:attribute>
➥<xsl:apply-templates/></div>
</xsl:template>

        <!-- we now make a special case of code as well, so we can
        ➥put in the link to copy the unadorned code to the clipboard -->
        <xsl:template match="code">
              <div class="code">
              <!-- right justify the button -->
              <table width="100%"><tr><td align="right">
                        <input type="button" value="Copy code"
                        ➥class="button">
                        <!-- when clicked, we want to call our (DHTML) script
                        ➥routine with the id of the appropriate code block -->
                        <xsl:attribute name="onclick">CopyCode('CodeBlock
                        ➥<xsl:eval>uniqueID(this)</xsl:eval>')
                        ➥</xsl:attribute>
                        </input>
              </td></tr></table>
              <xsl:apply-templates/>
              </div>
        </xsl:template>

<!-- The note element is a little special, because we want to put in a header
➥that says "Note" -->
<xsl:template match="note">
        <div class="note">
        <div class="noteHeader">Note</div>
        <xsl:apply-templates/>
        </div>
</xsl:template>

<!-- Titles for sections also map into their own div class -->
<xsl:template match="title">
        <div class="sectionHeader">
              <xsl:apply-templates/>
```

```
        </div>
</xsl:template>

<!-- underline elements are used to denote code in the text - turn these
➥into their own special spans -->
<xsl:template match="u">
        <span class="textcode"><xsl:apply-templates/></span>
</xsl:template>

<!-- handle vanilla HTML-like tags - just map them to the same thing -->
<xsl:template match="b|i|sub|sup|br">
        <xsl:element>
                <xsl:apply-templates/>
        </xsl:element>
</xsl:template>

<!-- spaceruns and tabs get mapped into sequences of  s.
➥We approximate tabs with 4 spaces -->
<xsl:template match="spacerun">
        <span><xsl:eval>Repeat(" ", this.getAttribute("len"))
        ➥</xsl:eval></span>
</xsl:template>

<xsl:template match="tab">
        <span><xsl:eval>Repeat(" ", 4)</xsl:eval></span>
</xsl:template>

<!-- translate special chars into entities for display in HTML - we use
➥numbered entities so we don't have to
        explicitly define them -->
<xsl:template match="char">
        <xsl:choose>
                <xsl:when test=".[@type = 'emDash']">&#8212;</xsl:when>
                <xsl:when test=".[@type = 'bullet']">&#8226;</xsl:when>
                <xsl:when test=".[@type = 'ellipsis']">&#8230;</xsl:when>
                <!-- we'll just use 2 nbsps for emSpace - #8195 doesn't seem
                ➥to work as documented -->
                <xsl:when test=".[@type = 'emSpace']">  </xsl:when>
                <xsl:when test=".[@type = 'smartApos']">&#146;</xsl:when>
                <xsl:when test=".[@type = 'smartLQuote']">&#147;</xsl:when>
                <xsl:when test=".[@type = 'smartRQuote']">&#148;</xsl:when>
                <xsl:otherwise><span class="unknown">char:
                ➥<xsl:value-of select="@type"/></span></xsl:otherwise>
        </xsl:choose>
```

```
</xsl:template>

<xsl:script>
<![CDATA[
// returns a string consisting of n copies of t
function Repeat(t, n)
{
	var r = "";
	while (n-- > 0)
		r += t;
	return r;
}

// format a codeline for copying (vs. display), eliminating all extra whitespace,
➥but putting in the space runs
// this is like a mini- apply-templates, but all in script
	function Expand(me)
	{
		var s = "";
		var children = me.childNodes;
		var c;

		// we walk thru all children of the codeline node
		while(c = children.nextNode())
		{
			switch(c.nodeName)
			{
				case "spacerun":
					// we want to expand these to
					➥the appropriate number of spaces
					s += Repeat(" ", c.getAttribute("len"));
					break;

				case "#text":
					// text nodes just get concatenated,
					➥but we want to get rid of all leading
					➥and trailing
					// whitespace, which the various
					➥transformations have added
					s+= Trim(c.nodeValue);
					break;

				default:
```

```
228:                         // flag any other nodes which
                             ➥might be in here
229:                         s+= "!" + c.nodeName + "!";
230:                 }
231:             }
232:             // remove leading line number
233:             s = s.replace(/^\d+:/, "");
234:             return s;
235:         }
236: 
237:         // remove any leading and trailing whitespace
238:         function Trim(s)
239:         {
240:             s = s.replace(/^\s*/, "");
241:             return s.replace(/\s*$/, "");
242:         }
243: ]]>
244: </xsl:script>
245: 
246: </xsl:stylesheet>
```

הקטעים המודגשים הם תחומים של גיליון הסגנונות, השונים ממה שראינו עד כה. מטרת כל אחד מהשינויים הללו היא לתמוך בטיפול ב-codeblock ב-HTML.

התוספת החשובה הראשונה היא בשורות 31-44 שם אנו מספקים את הפונקציה CopyCode. זהו הקוד שזה עתה בחנו. אנו מכניסים אותו לקובץ ה-HTML בדיוק כמו שעשינו עם כל התסריטים שבצד הלקוח: על ידי תחימתו בתגיות ה-HTML, <script>, המגנות עליו בעזרת הערות מדפדפנים מיושנים, ותחימה בקטע CDATA להגנה מפני מנתח תחביר ה-XSL. הסברנו זאת בפירוט בפרקים הקודמים, ולכן לא נרחיב על כך שוב.

אי ה-XML המחזיק את ה-codeData הממשי מוצג בשורות 74-92. אנו מכניסים רכיב xml בשורה 75, ומקודדים את התכונה id שלו ל-"theCodeData", כמצופה מתסריט בצד הלקוח.

בשורה 76 אנו מתחילים לולאת for-each של טרנספורמציית XSL. לולאה זו תעבור באיטרציות על הרכיבים התואמים: במקרה זה כל הרכיבים מטיפוס code בתחום הנוכחי. כזכור, הלוכסן הכפול מוצא את כל הצאצאים, לא רק את הבנים הישירים.

התחום המיידי של חיפוש זה הוא הרכיב הנוכחי, שבשורה 65 ניתן לראות כי זהו הרכיב story או book – כלומר, כל המסמך.

בשורה 77 אנו מגדירים רכיב codeblock שיחזיק כל קטע code שימצא. בשורה 79 אנו משייכים את התכונה id לרכיב codeblock הזה. הערך של התכונה הזו הוא המחרוזת codeBlock מלווה בתוצאת הקריאה לפונקציית ה-XSL, uniqueID. ל-uniqueID אנו מעבירים את הרכיב הנוכחי (הרכיב code המתאים).

UniqueID היא פונקציה קטנה וחזקה. היא מחזירה ערך אשר מובטח שהוא ייחודי לכל רכיב שנותח תחבירית במסמך. ההיבט החיוני של הפונקציה הזו הוא בכך שהיא מבטיחה להחזיר את **אותו** מזהה ייחודי בכל פעם שניתן לה רכיב מסוים. כך, שאם נקרא שוב ל-uniqueID עם אותו הרכיב, נקבל את אותו הערך בכל פעם מחדש. בהמשך הקובץ זה יהיה חשוב מאוד.

עדיין בתוך ה-codeblock שאנו יוצרים, בשורה 81 אנו קוראים ל-apply-templates. הפעם אנו מוסיפים הצהרת select, אך החיפוש הוא אחר * המוצאת את כל **הרכיבים** (אך לא את כל הצמתים).

אנו מבצעים התאמה לרכיבי codeLine פעמיים: פעם אחת כדי לאחסן את הקוד בתוך אי XML ללא עיצוב מספרי קוד, ובפעם השנייה לעיצוב עבור התצוגה.

בשורה 82 אנו מבצעים התאמה לכל רכיב מטיפוס codeLine ומעבירים אותו לשיטה Expand, הממקמת את התוצאה ברכיב. העבודה של Expand היא להסיר רווחים מיותרים, עוקבים ונגררים, תוך שימור הרווחים הפנימיים. את דרך היישום ניתן לראות בשורות 203-235.



התאמה זו ל-codeline בשורה 82 היא בתוך התחום של הרכיב apply-templates בשורה 81, אשר בעצמו נמצא בתחום של הלולאה for-each. ההתאמה השנייה ל-codeline היא בשורה 122, בתוך הרמה העליונה או בתחום ברירת המחדל של גליון הסגנונות.

בשורה 207 אנו מאתחלים את המחרוזת s מאפס, ובשורה 208 אנו מאחזרים את כל ה-childNodes מהרכיב שהועבר. אנו מצפים שה-childNodes היחידים יהיו צמתי טקסט (הטקסט הפנימי של הרכיב) או צמתי רווחים שיצרנו. אנו רוצים לשמר את הרווחים אך לסלק את כל הרווחים הלבנים,המובילים והנגררים האחרים.

כשהשיטה nodename נקראת על רכיב כלשהו, היא מחזירה את שם התגית של הרכיב הזה, וכשהיא היא נקראת על צומת טקסט היא מחזירה את המחרוזת #text.

בשורה 216, כשמתקבלת התאמה לרכיב spacerun, אנו מוסיפים את מספר הרווחים הנדרש למחרוזת. כשמתקבלת התאמה ל-#text, אנו מוסיפים את הטקסט למחרוזת שלנו, לאחר שאנו מעבירים אותו דרך השיטה trim שלנו, המוצגת בשורות 238-243. trim משתמשת בשיטה replace, שהיא שיטה של אובייקט המחרוזת של JavaScript, להחלפת תו במחרוזת בתו אחר. היא מחליפה רווחים מובילים ונגררים במחרוזת ריקה ("") ובכך מסירה את הרווחים.



replace מקבלת שני פרמטרים, המחרוזת המקורית ומחרוזת ההחלפה. ב-Java וב-JavaScript ביטויים רגולריים נתחמים על ידי תו לוכסן קדמי (/) במקום מרכאות כפולות.

התו ^ מציין את תחילת השורה. התו s\ מציין רווח, וה- * מציינת "אפס או יותר", כך ניתן לקרוא את **/s*\s^/** כ"מצא את כל תווי הרווחים מתחילת השורה עד שנתקלים בתו שאינו תו רווח" והפרמטר השני "" גורם להחליף את הרווחים האלו במחרוזת ריקה.

Reaplce השנייה, הנקראת בשורה 241, מבצעת את אותו הדבר, אלא שהפעם היא עובדת בסוף השורה ($) במקום בתחילתה.

בשורות 227-229 נשים לב כי אנו מוסיפים כרגיל, מלכודת לרכיבי childNodes בלתי צפויים (שאינם spacerun או text).

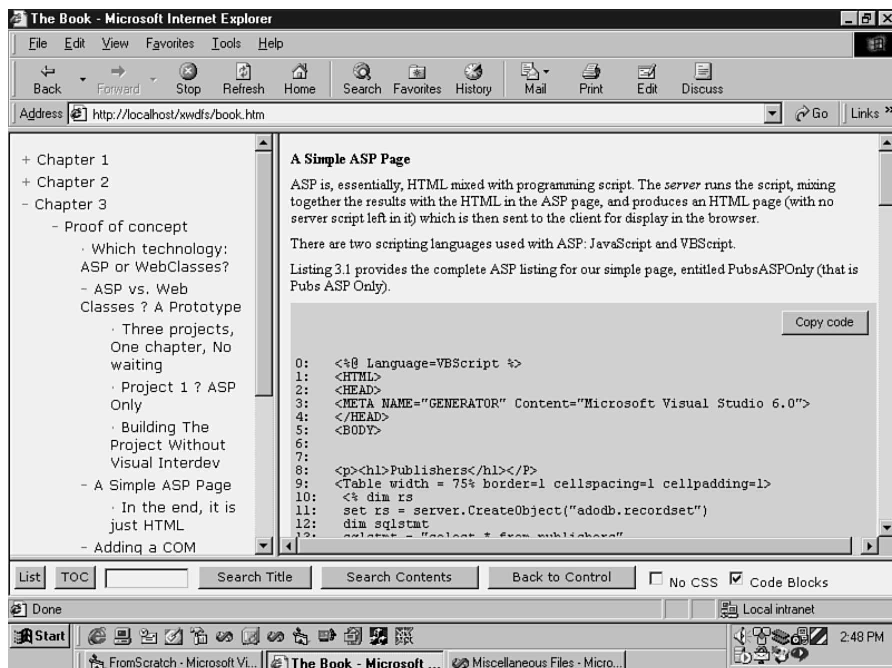
טקסט מורחב זה מוחזר בשורה 83 ומוכנס למסמך DOM הפלט. בשורה 84 אנו סוגרים את ההתאמה, בשורה 85 אנו סוגרים את apply-templates, ובשורה 86 את ה-codeblock. אנו חוזרים על כך עבור כל אחד מהרכיבים שנמצאו ולבסוף, בשורה 87, אנו סוגרים את הלולאה for-each. בשורה 88 התגית codedata נסגרת, כמו גם אי ה-XML עצמו.

לכן אנו מסוגלים עכשיו להכניס את ה-codedata לתוך ה-HTML יחד עם התסריט לשינויו. החתיכה היחידה החסרה היא, יצירת הכפתור בכל קטע קוד מוצג. את זה אנו עושים בשורות 126-138.

עד כה, לא היינו חייבים לקרוא לרכיב code ; טיפלנו בו בדיוק כמו בקטע note. עכשיו אנו מבצעים לו התאמה בנפרד בשורה 127. לכל רכיב code שנמצא אנו יוצרים בשורה 128 רכיב div חיצוני ואחר כך אנו יוצרים טבלה כך שנוכל להצמיד לימין את הכפתור. בשורה 131 אנו יוצרים את הכפתור ובשורה 133 אנו נותנים לכפתור תכונה עם האירוע onclick.

הנה החלק המתוחכם (והמעניין) : הליך הטיפול באירוע הוא הקריאה ל-CopyCode עם מזהה. אנו יוצרים את המזהה על ידי הוספת המחרוזת CodeBlock ולאחריה קריאה ל-uniqueID, תוך העברת הרכיב עצמו. כפי שצוין קודם לכן, מובטח לנו שיוחזר **אותו** מספר ייחודי כפי שהיה מוחזר בקריאה בשורה 79! זה יוצר את הקשר בין הכפתור ובלוק הקוד בסוף הקובץ.

כשאנו בוחנים את המקור של ה-HTML המופק על ידי גיליון ה-XSL, ניתן לראות שה-id המוצמד ל-codeblock יהיה בדיוק אותו id כמו זה המוצמד לכפתור המתאים! ניתן לבדוק זאת : נשתמש ביישום ליצירת הדף כמוצג בתרשים 8.10.



תרשים 8.10: עם בלוקי קוד.

בעוד זה מוצג בדפדפן שלנו, נפתח את המקור ונחפש אחר CodeBlock. נמצא את ה-HTML ליצירת הכפתור. עכשיו נחפש אחר ה-ID המוצג; נמצא אותו באי ה-XML שבסוף הקובץ.

מזל טוב, היישום גמור!

כאן סיימנו למעשה את היישום שאליו שאפנו להגיע. יצרנו יישום מלא לדפדוף בספר בחלקיו השונים, בצורה ידידותית למשתמש. תוך כדי העבודה, סקרנו את הטכנולוגיות החשובות המשתתפות בעבודה עם XML. ביניהם: XML חוקי ובנוי היטב, DTD, XSL וטרנספורמציות עבודה עם XML בצד השרת באמצעות טכנולוגיית ASP של "מיקרוסופט", ומסד נתונים SQL Server, שימוש בקבצי מערכת (File System Object), ואובייקטים של ADO. כמו כן, עבודה בצד הלקוח עם JavaScript ו-DHTML, HTML ו-CSS. וכמובן, שימוש באובייקטים שיצרנו בעצמנו עם VB.

ללא ספק, כמות חומר רבה, ושימוש במספר רב של כלים. מכך אתה למד, כמובן, כי עבודה יעילה עם קבצי XML ו-XSL כרוכה בשימוש נרחב עם כל הטכנולוגיות העכשוויות לבניית אתרים.

הבה נחשוב על זה לרגע, לקחנו ספר ופירקנו אותו אל תוך מסד נתונים, שמרנו אותו בפורמט XML ושיחזרנו אותו ליישום אינטרט.

כעת, נוכל לשלוף בקלות את הנתונים באופן המתאים לכל פלט שרק נרצה, למשל: מסמך PDF, הצגה ב-WAP, הצגה ב-Flash 5, הצגה ב-Microsoft Reader וכדומה.

זהו ייחודה של XML!

הצעדים הבאים

בפרק הבא נבחן יחידת שירות שנקראת XSL Helper - כשמה כן היא, באה להקל על העבודה עם גיליונות XSL. הפרק הבא מסביר כיצד ליצור תוכנית זו, תוך כדי חזרה על חלק מהחומר שעברנו בנושא XSL והצגה על גבי הדפדפן כ-HTML.

פרק 9
שימוש בתוכנת העזר XSL Helper ליצירה ותחזוקה של מסמכי XSL

בפרק זה:

- ביסוס יצירת XSL
- יישום XSL Helper
- הצגת XML עם HTML
- בחינת התוצאות
- עריכה ושמירה של שינויים
- הצעדים הבאים

בפרק זה נסתכל על XSL Helper, תוכנית עזר שנכתבה על ידי Mike Kraley כדי לסייע ביצירה ובתחזוקה של מסמכי XSL.

עזרה ביצירת XSL

XSL Helper נבנתה כדי לעזור ביצירה ובאיתור השגיאות בדפי XSL. חלק מן הקושי שביצירת דפי XSL הוא בניסיון לשמור על אינטראקציה בין ארבעה מסמכים שונים במקביל. ארבעת המסמכים הם:

1. מסמך ה-XML המקורי המשרת כמסמך המקור או הקלט.
2. גיליון הסגנונות ב-XSL שאנחנו יוצרים, ושמטרתו שינוי מסמך המקור בכדי להפיק את מסמך הפלט.
3. מסמך הפלט, שנוצר על ידי הפעלת ה-XSL על מסמך המקור.
4. ההצגה של מסמך הפלט (למשל, בדפדפן), באופן שבו הוא מוצג למשתמש, לאחר תהליך הטרנספורמציה.

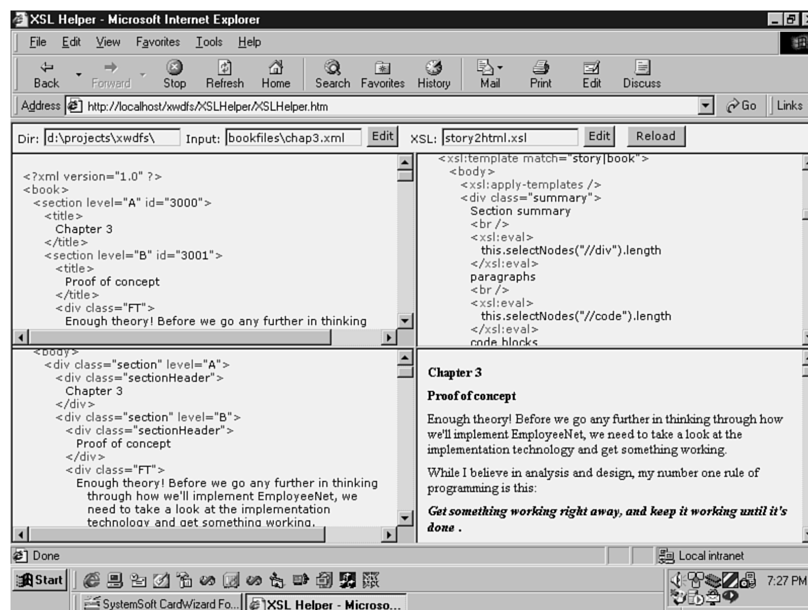
XSL Helper היא תוכנית עזר שמטרתה להציג בפניך סימולטנית את כל ארבעת המסמכים הללו. זוהי דוגמה טובה לאפשרויות העומדות בפניך כמתכנת, ליצירת עזרים משלך לעבודתך עם XML ו-XSL.

ממשק התוכנית מחולק לארבעה חלקים, שהם מסגרות HTML. בכל מסגרת מוצג אחד מבין המסמכים שמנינו לעיל.

נניח והיינו יוצרים קובץ XSL בשם : story2HTML.xsl, שתפקידו להמיר מסמך מ-XML ל-HTML.

לשם כך, היינו משתמשים במסמך קלט XML בשם chap3.xml ומסמך הפלט היה יכול להיות chap3.html.

XSL Helper היתה מציגה את chap3.xml במסגרת השמאלית העליונה, את story2HTML.xsl במסגרת הימנית העליונה, את chap3.xml, כקוד HTML, במסגרת השמאלית התחתונה, ואת chap3.xml, כפי שייראה בדפדפן, במסגרת הימנית התחתונה. כל זה מוצג בתרשים 9.1.



תרשים 9.1: XSL Helper

בנוסף לארבעת המבטים האלו, ישנה מסגרת עליונה קטנה, המכילה תיבות טקסט וכפתורים. כדי להשתמש ב-XSL Helper יש להכניס לתיבת הטקסט הראשונה את הנתיב לתיקיה שבה מאוחסנים אובייקטי ה-XML ודפי ה-ASP. בתיבת הטקסט השנייה יש להכניס את הקובץ בו נרצה להשתמש כקלט (בדוגמה המוצגת chap3.xml), ובתיבת הטקסט השלישית יש להכניס את גיליון הסגנונות שב-XSL, שפיתחנו או שאנו מעוניינים לבחון (בדוגמה המוצגת story2html.xsl).

לחיצה על לחצן Reload גורמת ל-XSL Helper לקרוא את קובץ הקלט ולהציג אותו במסגרת השמאלית העליונה. היא מציגה ייצוג HTML של קוד ה-XML הממשי, כפי שנתאר בקרוב. באופן דומה, XSL Helper קוראת את קובץ ה-XSL שסופק, ומציגה אותו במסגרת הימנית העליונה.

במקרה זה, התוצאה של טרנספורמציית ה-XSL היא מסמך HTML, המוצג במסגרת השמאלית התחתונה. לבסוף, ה-HTML המתקבל מובא לתצוגה במסגרת הימנית התחתונה בדיוק כאילו היה בדפדפן IE5.
הן קובץ הקלט והן קובץ ה-XSL יכולים להיערך **במקום** על ידי לחיצה על הכפתור Edit המתאים במסגרת העליונה.

בהמשך הפרק נסקור כיצד XSL Helper מיושמת. בדרך נבחן את עיבוד ה-XML שבצד הלקוח.



XSL Helper **דורשת לחלוטין** את IE5. לעתים האבטחה בתוך IE5 מגבילה את השימוש בקובץ זה. כדי להפחית זאת, יש לבחור בתפריט באפשרות Tools/InternetOptions **ולבחור** Security tab.

כש-XSL Helper נטענת, ניתן למצוא את האזור המופיע בפינה הימנית התחתונה של הדפדפן. במחשבים מסוימים, מופיע Local Intranet; במחשבים אחרים יתכן ומופיע אחרת. אנו עשויים להזדקק להגדרות אבטחה המותאמות אישית, שיאפשרו לכם את פקדי ה-ActiveX שבשימוש XSL Helper. נראה כי קביעה ל-Enable, בחלק מן המחשבים, פותרת את הבעיה, אם כי ידוע שקשה לבצע זאת נכון.

כמו כן, XSL Helper משתמשת באובייקט מערכת הקבצים. זו לא אמורה להיות בעיה, מכיון שכבר השתמשנו ב-VB6 במהלך התרגילים בספר, והתקנת VB מתקינה את אובייקט מערכת הקבצים.

יישום XSL Helper

נשים לב כי בתרשים 9.1 הקובץ הממשי שאליו ניווט הדפדפן הוא XSLHelper.htm. קובץ זה מכיל את קבוצות המסגרות, המכילות את ארבע המסגרות שהוצגו בתרשים 9.1. הקוד של קובץ זה מוצג בתדפיס 9.1.

תדפיס 9.1

```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" content="text/html; charset=iso-8859-1">
<TITLE>XSL Helper</TITLE>
</HEAD>
<frameset frameborder=1 border=0 rows="28,*">
  <frame name=Header src="XSLHHeader.htm">
  <frameset frameborder=1 rows="50%,50%" cols="50%,50%">
        <frame name=Input>
        <frame name=StyleSheet>
```

```
        <frame name=OutputSource>
        <frame name=Output>
  </frameset>
</frameset>
</HTML>
```

ישנן שתי קבוצות מסגרות בקובץ זה. קבוצת המסגרות החיצונית מכילה מסגרת בודדת, Header. קבוצת המסגרות החיצונית מכילה גם קבוצת מסגרות פנימית, אשר בעצמה מכילה ארבע מסגרות נוספות: Input, StyleSheet, OutputSource, ו-Output. אלו מתאימות למסגרות השמאלית העליונה, הימנית העליונה, השמאלית התחתונה, והימנית התחתונה, בהתאמה.

נשים לב כי לאף אחת מן המסגרות הללו אין תוכן פרט ל-Header, המציגה את XSLHeader.htm, המוצג בתדפיס *9.2*. מסמך זה הוא שמבצע את העבודה, ואנו נבחן אותו בפירוט.

תדפיס 9.2

```
<HTML>
<HEAD>
  <META HTTP-EQUIV="Content-Type" content="text/html; charset=iso-8859-1">
  <style>
  * {font-family:Verdana;font-size:8pt;}
  </style>
</HEAD>
<script>
 var fso = new ActiveXObject("Scripting.FileSystemObject");

 // this is the main function, called by pressing the Reload button
 function Refresh()
 {
        // remember our parameters in a cookie, so we don't have
        ➥to keep typing them in
        var cs = "Params=" + escape(Input.value) + "|"
        ➥+ escape(StyleSheet.value);
        cs += "|" + escape(Dir.value)
        var expDate = new Date();
        expDate.setFullYear(expDate.getFullYear()+1);
        cs += ";expires=" + expDate.toGMTString();
        document.cookie = cs;

        // clear out the contents of the four frames
        parent.frames.Input.document.clear();
        parent.frames.StyleSheet.document.clear();
        parent.frames.OutputSource.document.clear();
```

```
    parent.frames.Output.document.clear();
    // create XML documents for the input xml and the style sheet
    var XMLInput = new ActiveXObject("Microsoft.XMLDOM");
    var XMLStyleSheet = new ActiveXObject("Microsoft.XMLDOM");
    XMLInput.async=false;
    XMLStyleSheet.async=false;
    // parse the XML for the input and the stylesheet,
    ➥checking and reporting if errors
    XMLInput.loadXML(ReadFile(Input.value));
    if (XMLInput.parseError.reason)
        ReportParseError("input file", XMLInput);
    XMLStyleSheet.loadXML(ReadFile(StyleSheet.value));
    if (XMLStyleSheet.parseError.reason)
        ReportParseError("style sheet", XMLStyleSheet);
    // render the input files, transforming them with out
    ➥standard ShowXML.xsl to an HTML version
    var s = XMLInput.transformNode(XMLDisplayStyle.XMLDocument);
    parent.frames.Input.document.body.innerHTML = s;
    s = XMLStyleSheet.transformNode(XMLDisplayStyle.XMLDocument);
    parent.frames.StyleSheet.document.body.innerHTML = s;
    // now do the desired transform, and check for errors
    var XMLOutput = new ActiveXObject("Microsoft.XMLDOM");
    XMLOutput.async = false;
    XMLInput.transformNodeToObject(XMLStyleSheet, XMLOutput);
    if (XMLOutput.parseError.reason)
        ReportParseError("output", XMLOutput);
    // and render the output - once transformed into HTML and once as is
    s = XMLOutput.transformNode(XMLDisplayStyle.XMLDocument);
    parent.frames.OutputSource.document.body.innerHTML = s;
    parent.frames.Output.document.write(XMLOutput.xml);
}
// report the details of a parsing error
// call with a string that identifies the document and the DOM document itself
function ReportParseError(src, dom)
{
    var s = "Error parsing " + src + ": " + dom.parseError.reason;
    s += "\nline: " + dom.parseError.line;
    s += "\npos: " + dom.parseError.linepos;
```

```
		s += "\nsrc: " + dom.parseError.srcText;
		alert(s);
	}

	// allow the user to edit the contents of the specified frame
	function Edit(n)
	{
		// get a reference to the specified frame
		var d = parent.frames(n).document;

		// read the contents of the file as a string
		var s = ReadFile(document.all(n).value);

		// now set the contents of the frame to be one large <textarea>
		d.body.style.margin=0;
		d.body.innerHTML = "<textarea id=EditText style='width:100%;height:98%;
		➥border:0;font-family:Verdana;font-size:8pt;'>" + "</textarea>";

		// set the contents of the text area to the contents of the file
		d.all("EditText").value = s;

		// display the Save and Cancel buttons and hide the Edit button
		var prefix = n.substr(0,2);
		document.all(prefix + "Save").style.display="";
		document.all(prefix + "Cancel").style.display="";
		document.all(prefix + "Edit").style.display="none";
	}

	// read the contents of the specified file as a string
	function ReadFile(fn)
	{
		var f = fso.OpenTextFile(BuildPath(Dir.value, fn), 1);
		var s = f.ReadAll();
		f.Close();
		return s;
	}

	// save the edited file to disk
	function Save(n)
	{
		// open the appropriate file name for writing
		var f = fso.CreateTextFile(BuildPath(Dir.value, document.all(n).value), true);

		// and write the contents of the textarea
```

```
        f.Write(parent.frames(n).document.all("EditText").value);
        f.Close();

        // Cancel handles the common tasks to getting back to the pre-Edit state
        Cancel(n);
}

// user has pressed the cancel button - get back to the original state
function Cancel(n)
{
        // hide the Save and Cancel buttons, show the Edit button
        var prefix = n.substr(0,2);
        document.all(prefix + "Save").style.display="none";
        document.all(prefix + "Cancel").style.display="none";
        document.all(prefix + "Edit").style.display="";

        // and act as if Reload has been pressed - this will refresh all frames
        Refresh();
}

// create a file path from the given directory and file name
function BuildPath(d, f)
{
        return fso.BuildPath(d, f);
}
</script>

<script for=window event=onload>
// initialize the button visibility
InSave.style.display="none";
InCancel.style.display="none";
StSave.style.display="none";
StCancel.style.display="none";

// restore values from cookie
var crumbs = document.cookie.split(";");
if (crumbs[0])
{
        var nv = crumbs[0].split("=");
        if (nv[0] == "Params")
        {
                var cs = nv[1].split("|");
                Input.value = unescape(cs[0]);
                StyleSheet.value = unescape(cs[1]);
```

```
154:             Dir.value = unescape(cs[2]);
155:         }
156: }
157: </script>
158:
159:
160: <BODY style='margin:2 5;'>
161:
162: <!-- this XML island holds our standard "view XML as HTML" style sheet which
     ➥we reuse many times and don't have to change -->
163: <xml id="XMLDisplayStyle" src="ShowXML.xsl" ></xml>
164:
165: <!-- the UI -->
166: Dir: <input id=Dir>
167: Input: <input id=Input>
168: <input type=button id=InEdit value=Edit onclick="Edit('Input')">
169: <input type=button id=InSave value=Save onclick="Save('Input')">
170: <input type=button id=InCancel value=Cancel onclick="Cancel('Input')">
171:
172: XSL: <input id=StyleSheet>
173: <input type=button id=StEdit value=Edit onclick="Edit('StyleSheet')">
174: <input type=button id=StSave value=Save onclick="Save('StyleSheet')">
175: <input type=button id=StCancel value=Cancel onclick="Cancel('StyleSheet')">
176:
177: <input type=button id=Reload value=Reload onclick="Refresh()">
178: </BODY>
179: </HTML>
```

XSLHelper.htm מחולק לשני חלקים: הכותרת העליונה והגוף. הכותרת העליונה היא משורות 1 עד 159. הגוף מתחיל בשורה 160. אנו נתחיל את הניתוח שלנו בשורה 166.

בשורות 166-167 נוצרים הפקדים ומוצבים בהם שיטות הטיפול השונות. לשני שדות הטקסט Input ו-XSL מקושרים שלושה כפתורים (Edit, Save ו-Cancel). כשהדף נטען לראשונה, השיטה onLoad תופעל, והכפתורים Save ו-Cancel יוחבאו, כמוצג בשורות 139-142:

```
139: InSave.style.display="none";
140: InCancel.style.display="none";
141: StSave.style.display="none";
142: StCancel.style.display="none";
```

אם המשתמש לוחץ על Edit, אז הכפתורים Save ו-Cancel יוצגו. נראה זאת מאוחר יותר בפרק.

המשתמש מכניס את שמות הקבצים המתאימים (או שהם משוחזרים מעוגיות, נראה זאת בהמשך) ואחר כך לוחץ על Reload. הכפתור Reload נוצר בשורה 177, ומקושרת אליו השיטה Refresh, בה מבוצעת עיקר העבודה:

```
177:  <input type=button id=Reload value=Reload onclick="Refresh()">
```

מה מבצעת ()Refresh

התפקיד של Refresh הוא לנקות את המסגרות ואחר כך למלא אותן במסמכים כפי שתואר קודם. זאת נעשה על ידי יצירת מסמך DOM ב-XML עבור קובץ הקלט שב-XML, ומסמך DOM ב-XML אחר עבור גיליון הסגנונות. הן קובץ הקלט שב-XML והן גיליון הסגנונות שב-XSL (שהוא בעצמו קובץ XML) מנותחים תחבירית, לבדיקת שגיאות ודיווח על בעיות.

אנו רוצים להיות מסוגלים להסתכל על התוכן של קבצי ה-XML האלו במסגרות העליונות. הדרך הקלה ביותר לעשות זאת היא להמיר אותם לקבצי HTML ולהציג אותם על ידי שימוש בטכנולוגיית דפדפן בסיסית. כדי להשיג זאת, השיטה Refresh משתמשת ב-ShowXML.xsl. זהו גיליון סגנונות פשוט ב-XSL, שיצרנו להבאת קובץ XML לתצוגה בדפדפן. נבחן קובץ זה ביתר פירוט בעוד רגע.

כעת, משקובץ המקור ב-XML וגיליון הסגנונות ב-XSL הובאו לתצוגה, הצעד הבא הוא להשיג את הטרנספורמציה המוכתבת על ידי גיליון הסגנונות ב-XSL, לדווח על שגיאות שהתעוררו, ולהציג את קובץ הפלט המתקבל. למעשה, פלט זה מובא לתצוגה פעמיים: פעם אחת להצגת תגיות ה-HTML בגירסה המוצגת במסגרת השמאלית התחתונה, ופעם שנייה להצגת הפלט כפי שהיה נראה בדפדפן, במסגרת הימנית התחתונה.

כיצד Refresh עובדת

הצעדים של Refresh הם כדלקמן:

1. איסוף כל הפרמטרים שהוכנסו על ידי המשתמש ואחסונם בעוגיה.
2. ניקוי המסגרות.
3. הצגת קובץ הקלט וגיליון הסגנונות (ה-XML וה-XSL), במסגרות המתאימות להם.
4. ביצוע טרנספורמציית ה-XSL על קובץ XML הקלט.
5. הצגת קובץ הפלט המתקבל, הן כ-HTML והן כפי שהיה נראה למשתמש על הדפדפן.

איסוף הפרמטרים

כאמור, XSL Helper דורשת שלוש פיסות מידע, או "פרמטרים", והן: קובץ הקלט (chap3.xml), גיליון הסגנונות (story2HTML.xsl), וקובץ הפלט (chap3.html). לאחר מספר פעמים, תהליך הכנסת הערכים הללו, נעשה מייגע, ולכן אנו מאחסנים אותם

בעוגיות לשם הנוחות. במקום להתעסק עם אוספי העוגיות, בשורה 14 אנו פשוט משרשרים את שלושת פיסות המידע הללו לתוך מחרוזת אחת, ושומרים אותה כעוגיה בודדת.



escape() היא שיטה ב-JavaScript המקודדת מחרוזות טקסט כך שהן יוכלו להיות מאוחסנות בבטחה בתוך עוגיה (Cookie). ניתן להחזיר מחרוזות שהשיטה escape() הופעלה עליהן למחרוזות רגילות, על ידי שימוש ב-unescape(), כפי שמוצג מאוחר יותר בקובץ זה.

ואם נדייק עוד יותר: ערכה של עוגיה (Cookie) לא יכול להכיל רווחים. Escape() פשוט דואגת להחליף כל רווח בסימן 20% המייצג את תו הרווח, ובאחזור העוגיה בעזרת unescape() מוחזרת המחרוזת שוב עם רווחים.

בשורות 16 ו-17 תאריך אנחנו מכוונים את זמן התפוגה של העוגיה לעוד שנה מעכשיו, ובשורה 19 אוסף העוגיות של המסמך, נקבע להחזיק את העוגיה עם הפרמטרים של הקבצים:

```
19:    document.cookie = cs;
```

בשורות 22-25 אנו מנקים את כל המסגרות כהכנה להצגת ארבעת המבטים.

הצגת XML הקלט וגיליון ה-XSL

בשורות 28 ו-29 אנו יוצרים שני אובייקטים חדשים, שכל אחד מהם הוא מסוג XMLDom, אשר יחזיק XML DOM:

```
28:    var XMLInput = new ActiveXObject("Microsoft.XMLDOM");
29:    var XMLStyleSheet = new ActiveXObject("Microsoft.XMLDOM");
```

לאחר קביעת המאפיין שלהם, async ל-false (כפי שצוין בפרקים קודמים, זה גורם לתוכנית לעצור עד שהמסמכים נטענים במלואם), אנו מוכנים לטעון את מסמכי ה-XML מהקבצים המתאימים שלהם.

בשורה 34 אנו ממלאים את מסמך ה-DOM שב-XML הראשון עם מסמך הקלט (המקור):

```
34:    XMLInput.loadXML(ReadFile(Input.value));
```

זה מעורר את השיטה ReadFile(), המוצגת בשורות 95-100, אשר יוצרת אובייקט FileSystemObject חדש, קוראת את התוכן של הקובץ, ומחזירה את התוכן כמחרוזת.

הקריאה ל-loadXML גורמת למחרוזת להיקרא לתוך מסמך ה-XML DOM. אם יש שגיאות ניתוח תחבירי, אנו נדווח למשתמש על השגיאות האלו באמצעות השיטה ReportParseError, המוצגת בשורות 62-68.

שיטה זו, ReportParseError, יוצרת מחרוזת מהאובייקט parseError של מסמך ה-DOM, ומניחה את המחרוזת בתוך תיבת התראה מסוג alert. כדאי להקדיש רגע לסקירת השיטה הפשוטה הזו, שכן היא חושפת את הפרטים הזמינים בשבילנו, ממנגנון דיווח השגיאות המובנה במסמך ה-DOM:

```
64:        var s = "Error parsing " + src + ": " + dom.parseError.reason;
65:        s += "\nline: " + dom.parseError.line;
66:        s += "\npos: " + dom.parseError.linepos;
67:        s += "\nsrc: " + dom.parseError.srcText;
68:        alert(s);
```

בחזרה לשורה 37, אנו טוענים את גיליון ה-XSL באמצעות אותו המנגנון, ומדווחים על שגיאות ניתוח תחבירי עבור גיליון הסגנונות גם כן:

```
37:        XMLStyleSheet.loadXML(ReadFile(StyleSheet.value));
38:        if (XMLStyleSheet.parseError.reason)
39:              ReportParseError("style sheet", XMLStyleSheet);
```

עתה כאשר שני הקבצים נטענו, כל אחד במסמך ה-DOM שלו, אנו מוכנים להצגתם בדפדפן.

המטרה היא שהדפדפן יציג בדיוק את ה-XML, כפי שהוא יכול להיות מוצג בעורך הטקסט שלנו. אם היינו מנסים פשוט להזין את ה-XML לדפדפן, הוא היה מנסה לתרגם אותו; זה לא מה שאנו רוצים. במקום זאת אנו רוצים שהדפדפן יציג את ה-XML כתוכן, כמעט כמו תמונה של הקובץ.

כדי לעשות זאת, אנו צריכים שה-XML ייהפך למחרוזת HTML שתוכנס למסגרות המתאימות.

זוהי פעולה בת שני שלבים. ראשית, המסמך הופך למחרוזת בהתבסס על גיליון הסגנונות ShowXML.xsl, שנבחן בפירוט בעוד רגע. אחר כך המחרוזת עצמה תוכנס לתוך המסגרת. אנו מתחילים בשורה 42:

```
42:        var s = XMLInput.transformNode(XMLDisplayStyle.XMLDocument);
```

השיטה transformNode מקבלת גיליון סגנונות כארגומנט. כדי להבין את הארגומנט שאנו מספקים, אנו חייבים להסתכל על שורה 163:

```
163:  <xml id="XMLDisplayStyle" src="ShowXML.xsl" ></xml>
```

כאן, בגוף מסמך ה-HTML, הגדרנו **אי XML** שה-id שלו הוא XMLDisplayStyle ומסמך המקור שלו הוא ShowXML.XSL. כשאי ה-XML הזה נוצר, ShowXML.XSL נקרא מהקובץ ונוצר בזיכרון. אנו יכולים לגשת אליו כמסמך XML על ידי קריאה למאפיין XMLDocument, כפי שאנו עושים בשורה 42. מסמך ה-XML הזה מהאִי, הוא זה שמועבר כפרמטר לשיטה transformNode של XMLInput.

בקיצור, קראנו לשיטה transformNode על מסמך הקלט ב-XML (מקור) שלנו, תוך העברת גיליון הסגנונות ShowXML.XSL. גיליון הסגנונות הזה מציג את מסמך ה-XML הזה כ-HTML, כך שאנו יכולים להציג אותו במסגרת השמאלית העליונה. אנו נחזור

ל-ShowXML.XSL מאוחר יותר בפרק זה, ולעת עתה נניח שמוחזרת מחרוזת עם HTML חוקי להבאת מסמך המקור לתצוגה. מחרוזת זו ממוקמת במשתנה s, אשר מוצגת בשורה 43 במסגרת השמאלית העליונה:

```
43:		parent.frames.Input.document.body.innerHTML = s;
```

זה מושג על ידי קביעת המאפיין innerHTML השייך לרכיב הגוף של המסמך, מן המסגרת הפנימית.

אחר כך אנו חוזרים על תהליך זה בשורות 44-45 עבור גיליון הסגנונות, כך שהוא יוצג במסגרת הימנית העליונה.



עד עכשיו טענו את קבצי ה-XSL שלנו על ידי יצירת אובייקט מסמך DOM, וקריאה מפורשת לשיטות load או loadXML. כאן אנו משתמשים באִי XML, הן מכיון שזה נותן לנו את ההזדמנות להציג את הטכניקה הזו, והן מכיון שנשתמש בקובץ ה-XSL הזה לעתים תכופות, והרבה יותר יעיל לטעון אותו רק פעם אחת.

ביצוע הטרנספורמציה

השלב הבא שלנו הוא להחיל את גיליון הסגנונות שלנו על מסמך המקור ב-XML, ואנו עושים זאת בשורות 48-52. אנו מתחילים ביצירת מסמך פלט ב-XML בשורה 48:

```
48:		var XMLOutput = new ActiveXObject("Microsoft.XMLDOM");
```

לאחר קביעת המאפיין async שלו, ל-false, אנו קוראים לשיטה של מסמך המקור, transformNodeToObject, המקבלת שני ארגומנטים: שם גיליון הסגנונות עבור הטרנספורמציה, ושם האובייקט שבו אנו ממקמים את התוצאה:

```
50:		XMLInput.transformNodeToObject(XMLStyleSheet, XMLOutput);
```

בשורות 51 ו-52 אנו בודקים אם ישנן שגיאות ניתוח תחבירי, ואם אין כאלו, אז ה-XMLOutput מכיל עכשיו את התוצאה של החלת גיליון הסגנונות על מסמך המקור. כעת אנו מוכנים להצגת התוצאה הזו.



כאן השתמשנו ב-transformNodeToObject במקום ב-TransformNode. TransformNode מחזירה מחרוזת, ו-transformNodeToObject מקבלת כפרמטר השני שלה אובייקט, במקרה זה מסמך DOM, אליו היא ממקמת את תוצאות הטרנספורמציה.

מכיון שזו פעולה בת שני שלבים, יהיה נוח יותר לשמור את התוצאות בתוך אובייקט שנוכל לטפל בו אחר כך. בנוסף, אנו יכולים לבדוק באובייקט שגיאות ניתוח תחבירי, ואנו אכן עושים זאת בשורות 51 ו- 52.

הצגת התוצאות

בשורה 55, אנו יוצרים מחרוזת HTML עבור מסמך התוצאה, בדיוק כפי שעשינו קודם לכן עבור המקור וגיליון הסגנונות. בשורה 56 אנו מציגים מחרוזת זו בחלון השמאלי התחתון:

```
55:    s = XMLOutput.transformNode(XMLDisplayStyle.XMLDocument);
56:    parent.frames.OutputSource.document.body.innerHTML = s;
```

לבסוף, בשורה 57, אנו קוראים לשיטה write על המסגרת הימנית התחתונה, המביאה לתצוגה את ה-HTML שאנו מספקים לה, כאילו היה מוצג בדפדפן. ה-HTML שאנו מעבירים הוא התוצאה של גישה למאפיין xml על מסמך DOM הפלט. זה מחזיר את המחרוזת של XML, שבמקרה זה הוא HTML המוכן לתצוגה.

הצגת XML עם HTML

סקרנו באופן כללי, את אופן ההצגה בדפדפן של מסמך המקור שב-XML וגיליון הסגנונות שב-XSL. הבה נחזור אחורה, ונבחן זאת בפירוט יתר.

המשימה שלנו היא להפוך את ה-XML לתצורת HTML. באופן לא מפתיע, אנו נשתמש ב-XSL. אנו זקוקים לגיליון סגנונות שייצור HTML, כך שהדפדפן יציג את ה-XML בדיוק כפי שהיה מופיע אם היה מוצג בעורך טקסט. גיליון הסגנונות שבו נשתמש הוא ShowXML.XSL, כמוצג בתדפיס 9.3.

תדפיס 9.3

```
<?xml version="1.0"?>
<!-- XSL stylesheet to display arbitrary XML in an HTML browser - similar to,
  ➥but much simpler than, defaultss.xsl -->
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/TR/WD-xsl"
  xmlns="http://www/w3/org/TR/REC-html40"
  result-ns="">

<xsl:template match="/">
  <html>
      <body>
          <style>
                      * {font-family:'Verdana'; font-size:8pt;}
                  <!-- element container -->
                      .e  {margin-left:1em; text-indent:-1em;}
                  <!-- tag -->
                      .t  {color:#990000}
                  <!-- xsl: tag -->
                      .xt {color:#990099}
                  <!-- markup characters -->
```

```
                    .m  {color:blue}
            <!-- text node -->
                    .tx {color:black;}
            <!-- cdata -->
                    .di {font-family:Courier New;}
            <!-- DOCTYPE declaration -->
                    .d  {color:blue}
            <!-- pi -->
                    .pi {color:blue}
            <!-- comment -->
                    .ci {color:green}
        </style>
        <xsl:apply-templates/>
    </body>
 </html>
</xsl:template>

<!-- flag any unknown elements that make it this far -->
<xsl:template match="*">
 <div style="color:red"><xsl:apply-templates/></div>
</xsl:template>

<!-- attributes -->
<xsl:template match="@*" xml:space="preserve"><span class="t">
➥<xsl:node-name/></span><span class="m">="</span><xsl:value-of/>
➥<span class="m">"</span></xsl:template>

<!-- text nodes -->
<xsl:template match="textNode()">
 <div class="e">
 <span class="tx"><xsl:value-of/></span>
 </div>
</xsl:template>

<!-- CDATA -->
<xsl:template match="cdata()">
 <div class="e">
    <span class="m">&lt;![CDATA[</span>
        <span class="k"><pre><xsl:value-of/></pre></span>
    <span class="m">]]&gt;</span>
 </div>
</xsl:template>

<!-- comments -->
```

```
<xsl:template match="comment()">
 <div class="e">
      <span class="m">&lt;!--</span>
            <span class="ci"><xsl:value-of /></span>
      <span class="m">--&gt;</span>
 </div>
</xsl:template>

<!-- Doctype - can't read it, so just say that it's there -->
<xsl:template match="node()[nodeType()=10]">
<div class="e">
<span class="d">&lt;!DOCTYPE <xsl:node-name/><i> (Can't display...)
 ➥</i>&gt;</span>
</div>
</xsl:template>

<!-- PIs -->
<xsl:template match="pi()">
<div class="e">
<span class="m">&lt;?</span><span class="pi"><xsl:node-name/>
 ➥<xsl:for-each select="@*"><xsl:node-name/>="<xsl:value-of/>"
 ➥</xsl:for-each></span><span class="m">?&gt;</span>
</div>
</xsl:template>

<!-- Elements without descendants (leaves) - these get empty tags -->
<xsl:template match="*">
 <div class="e">
      <div>
            <span class="m">&lt;</span><span>
            ➥<xsl:attribute name="class"><xsl:if match="xsl:*">x
            ➥</xsl:if>t</xsl:attribute><xsl:node-name/></span>
            ➥<xsl:apply-templates select="@*"/><span class="m">
            ➥/&gt;</span>
      </div>
 </div>
</xsl:template>

<!-- Elements with any kind of descendants-->
<xsl:template match="*[node()]">
 <div class="e">
      <div>
            <span class="m">&lt;</span><span>
            ➥<xsl:attribute name="class">
```

```
        ➥<xsl:if match="xsl:*">x</xsl:if>t</xsl:attribute>
        ➥<xsl:node-name/></span>
        ➥<xsl:apply-templates select="@*"/><span class="m">&gt;</span>
97:       </div>
98:       <div><xsl:apply-templates/>
99:         <div>
100:          <span class="m">&lt;/</span><span>
          ➥<xsl:attribute name="class"><xsl:if match="xsl:*">x
          ➥</xsl:if>t</xsl:attribute><xsl:node-name/></span>
          ➥<span class="m">&gt;</span>
101:        </div>
102:      </div>
103:  </div>
104:  </xsl:template>
105:
106:  </xsl:stylesheet>
```

הדרך הטובה ביותר לבחון את הקובץ הזה היא מלמטה למעלה.

רכיבי עלים

הבה נתחיל בשורה 83, בה אנו עומדים לבצע התאמה לכל הרכיבים שאין להם צאצאים.



התאמת ה-XQL ‏(XML Query Language) היא ב- *. כיצד אנו יודעים שתתקבל התאמה רק לרכיבים ללא צאצאים? מכיון, שאנו מבצעים התאמה לכל הרכיבים עם צאצאים בשורה 93, כפי שנראה בעוד זמן קצר. נזכיר כי XSL עובדת מלמטה כלפי מעלה. לאחר שביצענו התאמה לכל הרכיבים עם צאצאים בשורה 93, רק הרכיבים שיהיו ללא צאצאים, יתאימו בשורה 84.

אנו מציבים div חיצוני ופנימי, החיצוני לשליטה בהזחה, והפנימי להחזקת התוכן שלנו.

הבה נפרק את שורה 87 לחלקים, שכן מתרחשים מספר דברים בשורת קוד בודדת זו:

```
87:  <span class="m">&lt;</span><span><xsl:attribute name="class">
     ➥<xsl:if match="xsl:*">x</xsl:if>t</xsl:attribute><xsl:node-name/></span>
     ➥<xsl:apply-templates select="@*"/><span class="m">/&gt;</span>
```

אנו מתחילים ברכיב span שתכונתו נקבעת ל-"class="m (גורם לטקסט להופיע בכחול). אחר כך אנו כותבים את התווים < שיופיעו בדפדפן כסוגר משולש פותח (<).

לבסוף, כמו בכל XML הבנוי היטב, הרכיב span נסגר:

```
87: <span class="m">&lt;</span><span><xsl:attribute name="class">
    ➥<xsl:if match="xsl:*">x</xsl:if>t</xsl:attribute>
    ➥<xsl:node-name/></span><xsl:apply-templates select="@*"/>
    ➥<span class="m">/&gt;</span>
```

אחר כך אנו יוצרים span חדש שהתכונה שלו היא "class="x או "class="xt, תלוי אם הרכיב הזה הוא ב-namespace של XSL.

אנו מגדירים האם יש לנו, או לא, רכיב ב-namespace של XSL באמצעות הרכיב xsl:if, אשר מפיק את הפלט שלו (במקרה זה האות x) רק אם תנאי המבחן הוא אמת. המבחן במקרה זה הוא התאמה של האותיות :xsl מלוות באפס או יותר אותיות. לאחר שזה מתבצע, אנו סוגרים את תגית התכונה, ומשתמשים בתגית ה-XML, node-name, הסוגרת את עצמה, כדי להעתיק את הצומת שזה עתה נמצא מתאים (דהיינו, הרכיב שנמצא מתאים בשורה 84), וסוגרים את המרווח:

```
87: <span class="m">&lt;</span><span><xsl:attribute name="class">
    ➥<xsl:if match="xsl:*">x</xsl:if>t</xsl:attribute><xsl:node-name/></span>
    ➥<xsl:apply-templates select="@*"/><span class="m">/&gt;</span>
```

בשלב הבא, אנו קוראים ל-apply-templates כדי לבצע התאמה לתכונות של הצומת. זה מטופל על ידי שורות אחרות בקובץ-XSL, שנבחן בעוד זמן קצר:

```
87: <span class="m">&lt;</span><span><xsl:attribute name="class">
    ➥<xsl:if match="xsl:*">x</xsl:if>t</xsl:attribute><xsl:node-name/></span>
    ➥<xsl:apply-templates select="@*"/><span class="m">/&gt;</span>
```

לבסוף, אנו מסיימים ברכיב span נוסף, "class="m, המחזיק את התגית הסוגרת שאנו יוצרים באמצעות /> המתורגם ב-HTML ל- />. רכיבי ה-div נסגרים בשורות 88 ו-89, והתבנית עצמה נסגרת בשורה 90.

עכשיו הוצאנו פלט של התגית השלמה ב-HTML בשביל רכיב העלה, כך שכשהיא מוצגת בדפדפן, היא תיראה בדיוק כמו שה-XML נראה בעורך.

רכיבים עם צאצאים

כעת אנו מוכנים לבחון את ההתאמה לרכיבים **שיש** להם צאצאים, כפי שנראה, החל משורה 92. ההיבט המעניין ביותר של תבנית זו, הוא הצהרת ההתאמה:

```
93: <xsl:template match="*[node()]">
```

לא ראינו שימוש בסוגריים מרובעים, כמו כאן, קודם לכן. הם נקראים בשם פילטר (filter), להלן הדרך להבנתם. אם היינו כותבים "match="foo/bar, XSL היתה מוצאת כל רכיב foo עם רכיב בן bar, ומחזירה את הרכיב bar. אם, לעומת זאת, היינו כותבים "[match="foo[bar אז XSL היתה מוצאת כל רכיב foo עם רכיב בן bar ומחזירה את הרכיב foo. כלומר, משמעות הדבר היא, "מצא את הרכיבים שיש להם בנים בפילטר".

* מצא התאמה לכל רכיב ו-()node המחזירים צומת כלשהי, כך שאת ההתאמה הזו יש לקרוא "מצא את כל הרכיבים שיש להם צומת כלשהי כבן" – כלומר, "מצא את כל הרכיבים שאינם עלים".

שוב אנו יוצרים div חיצוני ופנימי. למעשה, שורה 96 היא כמעט בדיוק כמו שורה 87, פרט לכך שהפעם אנו לא יוצרים תגית הסוגרת את עצמה; אלא תגית פותחת.

בהמשך, אנו יוצרים div שני שיכיל את התוכן (כלומר, את הבנים) של הרכיב שזה עתה נמצא מתאים. לבסוף, בשורות 99-101 אנו יוצרים div שלישי שיכיל את התגית הסוגרת.

התאמה להוראות עיבוד

בשורה 77 אנו מבצעים התאמה להוראות עיבוד, ומסמנים אותן לתצוגה בכחול. הבה נבחן את שורה 79 ביט אחר ביט, שכן תבנית זו חוזרת עבור רכיבים אחרים גם כן:

```
79:  <span class="m">&lt;?</span><span class="pi"><xsl:node-name/>
     ➥<xsl:for-each select="@*"><xsl:node-name/>="<xsl:value-of/>"
     ➥</xsl:for-each></span><span class="m">?&gt;</span>
```

אנו מתחילים עם מרווח שהמחלקה שלו נקבעת ל-"m" (עבור תווי סימון, Markup Characters) ולכן יוצג בכחול. בתוך זה אנו ממקמים ?<, שיוצג ב-HTML כ- <?:

```
79:  <span class="m">&lt;?</span><span class="pi"><xsl:node-name/>
     ➥<xsl:for-each select="@*"><xsl:node-name/>="<xsl:value-of/>"
     ➥</xsl:for-each></span><span class="m">?&gt;</span>
```

אנו ממשיכים במרווח שהמחלקה שלו נקבעת ל-"pi" (Prcessing Instruction) וגם זה יהיה בכחול. הרכיב הראשון שאנו רואים במרווח זה הוא <xsl:node-name/>, הרכיב סוגר את עצמו, וממקם את שם הצומת הנוכחי בתוך המרווח. לכן, אם נמצאה התאמה להוראת העיבוד הראשונה בקובץ:

```
<?xml version="1.0"?>
```

התוצאה של <xsl:node-name/> תהיה כתיבת xml לתוך תגית ה-span של html. אחר כך אנו מתחילים לולאת xsl:for-each, ועוברים באיטרציה על כל התכונות (select="@*"). לכל תכונה שנמצאת אנו מוצאים את השורה הבאה:

```
<xsl:node-name/>="<xsl:value-of/>"
```

המשמעות היא, מקם את שם הצומת, כשאחריו סימן השווה (=), ומימין לו, ערך התכונה. שוב, אם אנו מוצאים את הרכיב:

```
<?xml version="1.0"?>
```

זה יגרום לכך שהמחרוזת "version="1.0 תיכתב לתוך זרם ה-HTML. כשהתכונות נמצאות במקום, מרווח זה מסתיים:

```
79: <span class="m">&lt;?</span><span class="pi"><xsl:node-name/>
    ➥<xsl:for-each select="@*"><xsl:node-name/>="<xsl:value-of/>"
    ➥</xsl:for-each></span><span class="m">?&gt;</span>
```

ה-span האחרון יוצר את הסוגֵר הסוגר, כשלפניו משורשר סימן שאלה.

שוב, אם אנו מוצאים את הרכיב:

```
<?xml version="1.0"?>
```

ה-HTML שנפיק כתוצאה משורה 79 יהיה:

```
<div class="e">
  <span class="m">
    <?
  </span>
  <span class="pi">
    xml version="1.0"
  </span>
  <span class="m">
    ?>
  </span>
</div>
```

תוצאת ההצגה של ה-HTML בדפדפן, תהיה השורה:

```
<?xml version="1.0"?>
```

בדיוק כפי שקיווינו!

התאמה לרכיבים אחרים

בשורה 69 אנו עושים ל-DocType את מה שזה עתה עשינו להוראת העיבוד. כאן ההתאמה היא גם כן מעניינת. במקום לבצע התאמה לכל רכיב שיש לו צומת בן, מבוצעת התאמה לכל רכיב שיש לו צומת בן מטיפוס 10: כלומר, DocType.

NodeType, הוא ה-NodeType שב-XML DOM, המוגדר על ידי מניית ה-DOMNodeType. זוהי פשוט קבוצה של שמות וערכים (קבועים) המוצבים כחלק מסטנדרט ה-XML. ערך המנייה (enumerated value) עבור הצהרת ה-Document Type הוא 10.

בשורה 61 מתבצעת התאמה להערות XML ובשורות 63-66 הן נכתבות החוצה כהערות HTML.

בשורות 51-58 אנו מבצעים התאמה ומציגים קטעי CDATA. נשים לב, כי כאן תחמנו את ה-CDATA הממשי בתגיות <pre> ו- </pre>, כך שהדפדפן לא ינסה לשנות את התצורה הקיימת.

בשורה 44 מתבצעת התאמה לכל רכיבי ה-textNode והם נכתבים החוצה. כאן אנו פשוט תוחמים את הטקסט בתגיות span עם תכונת המחלקה "tx".

התאמה לתכונות

בשורה 42 מבוצעת התאמה לתכונות (כפי שהבטחנו שנעשה קודם לכן). הבה נפרק את השורה הזו:

```
42: <xsl:template match="@*" xml:space="preserve"><span class="t">
    ➥<xsl:node-name/></span><span class="m">="</span><xsl:value-of/>
    ➥<span class="m">"</span></xsl:template>
```

אנו מתחילים בהוראה האומרת למנתח תחביר ה-XSL לשמור את הרווחים בין תכונות:

```
42: <xsl:template match="@*" xml:space="preserve"><span class="t">
    ➥<xsl:node-name/></span><span class="m">="</span><xsl:value-of/>
    ➥<span class="m">"</span></xsl:template>
```

אנו יוצרים תגית span שתחזיק את שם התכונה. למשל, אם התכונה היא "foo="bar תגית זו תכיל foo:

```
42: <xsl:template match="@*" xml:space="preserve"><span class="t">
    ➥<xsl:node-name/></span><span class="m">="</span><xsl:value-of/>
    ➥<span class="m">"</span></xsl:template>
```

אחר כך אנו יוצרים תגית span נוספת שתחזיק רק את סימן השוויון ואת המרכאות הפותחות:

```
42: <xsl:template match="@*" xml:space="preserve"><span class="t">
    ➥<xsl:node-name/></span><span class="m">="</span><xsl:value-of/>
    ➥<span class="m">"</span></xsl:template>
```

הרכיב xsl:value-of כותב החוצה את הערך (למשל, bar) ואנו יוצרים תגית span אחרונה שתחזיק את המרכאות הסוגרות.

ראש גיליון הסגנונות

בשורה 37 נמצאת ההתאמה שלנו לרכיבים לא מוכרים. אנו מאמינים ומקווים שלעולם לא נגיע לכאן (אנו מבצעים כבר התאמה לכל הרכיבים, מאוחר יותר בקובץ) אך שמנו זאת כאן, כדי להוכיח את ההנחה שלנו (כי מתכנתים טובים חוגרים חגורה **וגם** גורבים ביריות).

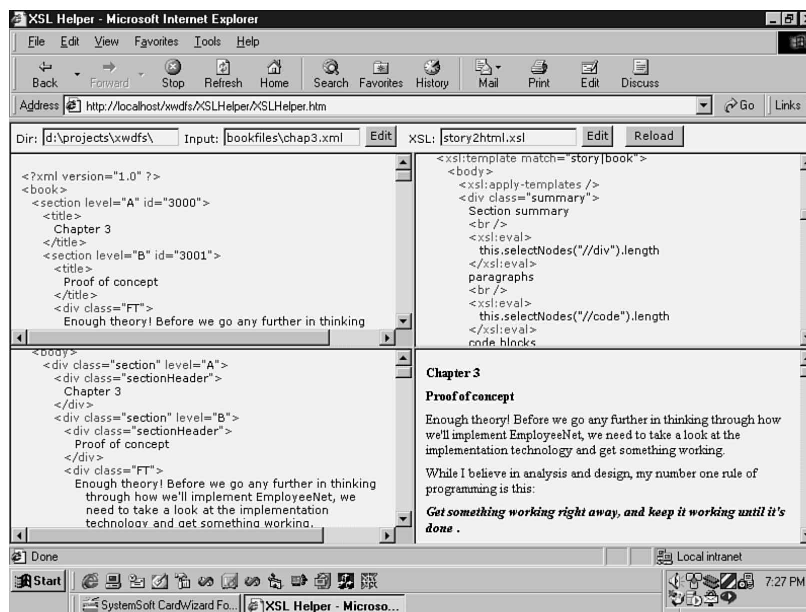
בשורה 7 מותאם רכיב השורש, וכאן אנו יוצרים את הרכיבים <html>, <body>, <style>, ומספקים את הסגנונות שישמשו במהלך מסמך ה-HTML המתקבל.

בחינת התוצאות

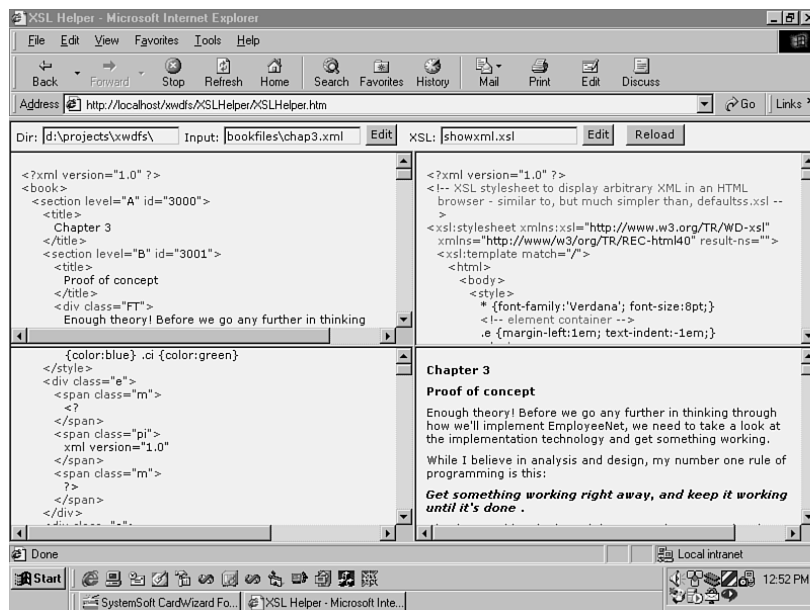
אם נתייחס לתדפיס 9.2, נמצא כי למעשה אנו מסוגלים להציג את קובץ ה-XML במסגרת השמאלית העליונה, בתוך הדפדפן. זוהי התוצאה הנראית של הקריאה ל-ShowXML.xsl על המסגרת הזו.

איך אנו יודעים כיצד נראה ה-HTML החבוי? מכיון שדפים אלו נטענים דינמית, אנו לא יכולים פשוט לבחון את המקור. אם נבחר באפשרות View Source בדפדפן, נראה רק את המקור של קבוצת המסגרות. אם נלחץ על הכפתור הימני של העכבר במסגרת השמאלית העליונה ונבחר ב-"view source", התוצאות הן אכן מאכזבות – ניתן לראות רק את התגיות <html>. נזכיר כי מסגרת זו מולאה באופן דינמי, על ידי שימוש ב-innerHTML.

ובכל זאת יש לנו פתרון. אנו יכולים להזין את chap3.xml לתוך XSL Helper, ולבקש ממנה לעבד את הקובץ, לא עם story2html.xsl, אלא עם ShowXML.xsl, כפי שמוצג בתרשים 9.3.



תרשים 9.2: קובץ ה-XML.



תרשים 9.3: שימוש ב-ShowXML.xsl.

ניתן לראות, כי chap3.xml עדיין מוצג בחלון השמאלי העליון. החלון הימני העליון מציג עכשיו את ShowXML.xsl כפי שהיינו מצפים. אבל, עיקר העניין הוא בחלון השמאלי התחתון. זהו ה-HTML שנוצר על ידי ShowXml.xsl – אותו HTML שנוצר על מנת ליצור את התצוגה בחלון השמאלי העליון!

אם תביט מקרוב תבחין שאחרי תגית ה-style, ממוקם ה-div החיצוני ("class="e), ה-div הפנימי ("class="m), ואחר כך ישנה תגית span ‏("class="m) עם התווים ?>. אחר כך תמצא תגית span נוספת ("class="pi) עם האותיות "xml version="1.0.

התוצאה הינה בדיוק כפי שחזינו מקריאת ה-XSL!



כעת משסיימנו עם כל הרקורסיה הזו, ננסה להתמודד עם הבעיה הבאה: כיצד אנו מביאים לתצוגה את ה-HTML המוצג במסגרת השמאלית התחתונה בתרשים 9.2? ככלות הכל, אם נציג רק את הקובץ כפי שהוא כתוב, לא נראה את התגיות. למעשה, זה מה שמוצג במסגרת **הימנית** העליונה.

אם כך, כיצד משיגים את ה"קסם" הזה? בקלות. ה-HTML שהפקנו הוא XML חוקי, כך שאנו מוסרים את התוכן (המופק על ידי ShowXML.xsl) מיד בחזרה ל-ShowXML.xsl. זה **יוצר** את ה- HTML שיציג את הדף הזה בתצוגה כפי שאנו רואים אותו!

עריכה ושמירה של שינויים

רצינו ש-XSL Helper תתמוך בעריכה גם של גיליון ה-XSL וגם של מסמך XML הקלט. כדי לראות כיצד יישמנו זאת, נחזור לשורות 71-92 של תדפיס 9.2, שם הוגדרה הפונקציה Edit, ומוצגת כאן שוב כתדפיס 9.4.

תדפיס 9.4

```
71: // allow the user to edit the contents of the specified frame
72: function Edit(n)
73: {
74:         // get a reference to the specified frame
75:         var d = parent.frames(n).document;
76:
77:         // read the contents of the file as a string
78:         var s = ReadFile(document.all(n).value);
79:
80:         // now set the contents of the frame to be one large <textarea>
81:         d.body.style.margin=0;
82:         d.body.innerHTML = "<textarea id=EditText style='width:100%;height:98%;
            ➥border:0;font-family:Verdana;font-size:8pt;'>" + "</textarea>";
83:
84:         // set the contents of the text area to the contents of the file
85:         d.all("EditText").value = s;
86:
87:         // display the Save and Cancel buttons and hide the Edit button
88:         var prefix = n.substr(0,2);
89:         document.all(prefix + "Save").style.display="";
90:         document.all(prefix + "Cancel").style.display="";
91:         document.all(prefix + "Edit").style.display="none";
92: }
```

הפרמטר מועבר כשהלחצן נלחץ, כפ שהוצג בשורות 168 ו- 173 בתדפיס 9.2:

```
168: <input type=button id=InEdit value=Edit onclick="Edit('Input')">
173: <input type=button id=StEdit value=Edit onclick="Edit('StyleSheet')">
```

כך, הערך של n יהיה Input או StyleSheet. אנו משתמשים בזה כדי לאחזר את המסמך במסגרת המתאימה, ומאחסנים זאת במשתנה המקומי d, בשורה 75:

```
75:         var d = parent.frames(n).document;
```

אחר כך אנו קוראים את התוכן של הקובץ כמחרוזת, תוך שימוש בשיטה ReadFile, המוצגת בתדפיס 9.2. בשורות 81-82 אנו הופכים את התוכן של המסגרת לאזור טקסט יחיד:

```
81:    d.body.style.margin=0;
82:    d.body.innerHTML = "<textarea id=EditText style='width:100%;height:98%;
       ➥border:0;font-family:Verdana;font-size:8pt;'>" + "</textarea>";
```

ובשורה 85 אנו משתמשים ב-DHTML כדי לקבוע את הערך (התוכן) של אזור הטקסט למחרוזת שקראנו מהקובץ:

```
85:    d.all("EditText").value = s;
```

כל מה שנותר לעשות הוא להחביא את הכפתור Edit ולהציג את הכפתורים Save ו-Cancel; זה מבוצע בשורות 89-91.

כאשר המשתמש שומר את הקובץ, הוא נכתב לדיסק, ובין אם הוא נשמר או מבוטל, אנו מחביאים את הכפתורים Save ו-Cancel, ומציגים את הכפתור Edit.

הצעדים הבאים

בזאת מסתיים מסענו בנושא XML ו-XSL. בפרק הבא נערוך סיכום קצר של הטכניקות בהן השתמשנו במהלך הלימוד, ונציג מקורות להמשך הלימוד. תודה על שהצטרפתם למסע, ואנא זכרו להישאר ישובים עד שהספר מגיע לסיומו המוחלט.

פרק 10
סקירת טכנולוגיות וטכניקות

בפרק זה:

- מבט לאחור
- עדיין לא ראינו דבר
- הצעדים הבאים

בפרק קצר זה, נסקור את הטכנולוגיות והטכניקות ששימשו ליצירת BiblioTech ו-XSL Helper, ונספק לכם הפניות והמלצות של מקורות להמשך הלמידה. XML היא סטנדרט מרגש, המתפתח במהירות, ובעוד שלא קיים ספר היכול להישאר עדכני במאה אחוזים, לאורך זמן, אנו משוכנעים שהחומר המסופק לכם בספר זה יהיה רלוונטי, ואנו מקווים, שגם מועיל במשך שנים רבות נוספות.

מבט לאחור על מה שעשינו

למרות ש-XML היא שיטה לבניית מידע לאחסון והחלפה, עבודה עם XML, בעצם עוסקת בטרנספורמציות. אם ניקח בחשבון את העבודה שהושקעה ביצירת BiblioTech, הרי כמעט כל מה שעשינו היה כרוך בהעברת המידע מתצורה אחת לאחרת. כוחן של XML ו-XSL הוא בכך שהן הופכות את הביצוע לקל מאוד.

במהלך בניית האפליקציות שלנו, השתמשנו בטכניקות רבות ושונות כדי לבצע את הטרנספורמציות הללו. זה כמובן היה מכוון. רצינו להראות את גמישותם של הכלים האלו, על ידי ביצוע הדברים בדרכים שונות ומגוונות. באפליקציה מסחרית ייתכן והיינו מנסים להיות יותר עקביים ביישום הטכניקה, אך מטרת הספר היא להדגים שיטות רבות ושונות, בתקווה שזה ייתן בידי הקורא את הכלים לבחור בטכניקה הטבעית וההולמת ביותר, ברגע האמת.

מעבר מ-HTML ל-XHTML

את הטרנספורמציה הראשונה שלנו, לא אנחנו כתבנו, ואנחנו לא יכולים אפילו לסקור אותה, אלא בעקיפין. כזכור, התחלנו כשבידינו מסמכי Word. רצינו לשמור את המידע שבמסמכי ה-Word באופן כזה שיאפשר לנו גמישות עם אותם הנתונים, כשנרצה להציגו במדיות שונות, ובפרט על גבי הדפדפן. ניצלנו את היכולות החדשות של Word 2000 בשמירת קובץ Word כקובץ XHTML. כפי שגילינו, מסמך XHTML זה, אומנם מסוג HTML, והוא אכן XML - אך אינו "בנוי היטב" (Well Formed) ואינו תואם את צרכינו. תצורה זו הסתכמה ל-HTML עם קצת XML מעורב. בעוד שזה לא הוביל אותנו לאן שרצינו להגיע, זה היה צעד ראשון הכרחי, וחסך לנו כמות נכבדת של תכנות.

מאותה נקודה המשכנו לקראת יצירת מסמך XML חוקי ובנוי היטב אשר יתאים לצרכינו, ויתאים ל-DTD משלנו. ראשית הפכנו את קובץ הפלט שקיבלנו מ-Word ל-XHTML כדי שנוכל להשתמש בכלים של ה-XML. כתבנו קוד VB מותאם אישית, כדי להפוך את הקובץ לתצורת XHTML – עדיין כמסמך HTML חוקי, אך גם כמסמך הבנוי היטב בתצורה של XML.



אם נסתכל רמה אחת מטה, בתהליך הטרנספורמציה מ-HTML ל-XHTML, נראה כי הוא למעשה מורכב משני טרנספורמציות.

ראשית, השתמשנו במנתח התחביר MSHTML של IE כדי להמיר HTML לייצוג אובייקטים בזיכרון. אחר כך, רכיב ה-VB המותאם אישית שלנו סרק את עץ האובייקטים שבזיכרון זה, והמיר דברים חזרה לייצוג ה-XHTML.

אנו רואים את התבנית הזו לעתים קרובות. אחד היתרונות של XML (ושל HTML) הוא בכך שהייצוג שלה על הדיסק, מופיע בתצורה הקריאה לבני אדם, אך הרבה יותר נוח ויעיל, לטפל במבנה שבזיכרון באמצעות עץ האובייקטים.

מעבר מ-XHTML ל-XML

משביצענו זאת, היה בידינו XML (XHTML) מתוקן - בנוי היטב. אך המסמך היה עדיין HTML בבסיסו: מכוון לתצוגה על גבי הדפדפן, ועם מבנה פנימי מרומז ברובו, בעוד שאנו זקוקים למסמך שאינו מכיל נתונים לגבי תצוגה, אלא רק נתונים לגבי תוכן. בנקודה זו רצינו להפוך את המסמך לתצורת ה-canoncial XML שלנו, שהיא הדרך בה אחסנו את התוכן, וממנה נבעו כל הגרסאות האחרות שלנו.

טרנספורמציה זו הושגה למעשה באמצעות שימוש בשלוש טכניקות. ראשית, ניצלנו כלי נפוץ מאוד: גיליון סגנונות XSL, בעזרתו הפכנו גירסה אחת של XML לגירסה אחרת. זה איפשר לנו להיפטר מחלקים ב-HTML שלא היינו צריכים יותר, ולשנות את התגיות והמבנה הייחודיים ל-HTML - לתצורה שאליה שאפנו.

אחר כך החלנו שתי טרנספורמציות אחרות, על ידי שימוש בקוד VB שיצרנו, בעבודה עם XML DOM טעון בזיכרון. אחת מהן העתיקה כל רכיב מעץ המקור אל מסמך DOM חדש, מיקמה אותו במבנה הנדרש, ואחר כך, נפטרה ממסמך ה-DOM המקורי. האחרת ארגנה מחדש את המבנה תוך העברת רכיבים בתוך מסמך ה-DOM עצמו.

משימתנו הבאה, היתה לפצל את המסמכים, מגודל של פרקים, ל"סיפורים" שהם היחידות האטומיות, אותם נאחסן במסד הנתונים. כאן קראנו שוב את ייצוג המחרוזת ב-XML לתוך מסמך DOM טעון בזיכרון. הפעם לא שינינו את המקור, אלא רק יצרנו מספר מסמכי DOM – אחד לכל סיפור – שניבנו על ידי העתקת רכיבים מהמקור. כשבנינו כל אחד, השתמשנו בשיטה xml. כדי להוציא גרסת מחרוזת של ה-XML לשמירה במסד הנתונים.

שימוש ב-XML ו-XSL להצגת הסיפורים

לאחר שיצרנו את הסיפורים, ושמרנו אותם על פי הירכיית הרכיבים שלהם במסד הנתונים. רצינו להציגם ויזואלית על גבי הדפדפן. לכן יצרנו גיליון סגנונות XSL חדש, שהמיר את ה-XML של הסיפור ל-HTML הניתן לתצוגה באמצעות הדפדפן. מחרוזת ה-XML אוחזרה ממסד הנתונים ונטענה באמצעות ()loadXML לתוך מסמך DOM חדש מתאים. ה-XSL נטען ישירות מהקובץ שלו באמצעות ()load. כתוצאה מכך, קיבלנו HTML שהוחזר כמחרוזת ל-ASP, אשר כתב אותו באמצעות Response. שוב, זוהי תבנית נפוצה במקרה של הצגת XML.

כדי לבנות את טבלת תוכן העניינים, נקטנו בקו פעולה אחר. במקום לעבוד מקובץ או אובייקט XML קיימים, בנינו מסמך XML על ידי שרשור מחרוזות בהתבסס על תוכן מסד הנתונים. וכך שרשרנו רכיב אחר רכיב לתוך מחרוזת. בביצוע שלנו השתמשנו בקוד ASP; בהפקה של אתר מסחרי, ייתכן והיינו משתמשים בפרוצדורה המאוחסנת במסד הנתונים עצמו, או באובייקט נפרד. ברגע שבנינו את מחרוזת ה-XML, השתמשנו בתבנית המוכרת של הפיכתה ל-HTML באמצעות גיליון סגנונות XSL.

כדי להדגים עוד יותר את הגמישות של XML, השתמשנו אחר כך בשתי טכניקות אחרות כדי להפיק את אותה טבלת תוכן עניינים. באפשרות הראשונה, שחזרנו בדיוק את אותו מסמך XML, הפעם לא על ידי שרשור מחרוזות אלא בעזרת טרנספורמציית XSL על קובץ הפרק שב-XML. ה-XML שהתקבל היה זהה ל-XML שהופק ממסד הנתונים, וכך היינו מסוגלים להשתמש באותו גיליון סגנונות ב-XSL כדי להפוך אותו ל-HTML.

הטכניקה השנייה היתה לשלב את שני מעברי ה-XSL לאחד. כאן יצרנו גיליון סגנונות ב-XSL שעבר ישירות מקובץ הפרק שב-XML ל-HTML המבוקש.

אחר כך ביצענו שינויים לגבי אופן הצגת הסיפורים - אחת עבור דפדפנים פשוטים, ואחרת שהגדילה את היכולות של הפלט, כדי שהמשתמש יוכל לבצע פעולות של העתק/גזור והדבק על טקסט. זה הראה כיצד, פשוט על ידי שינוי גיליון סגנונות XSL, אנו מסוגלים להפיק פלטים שונים שיתאימו לצרכים שונים. וזה הרי כל הרעיון!

XSL בצד הלקוח

עד לנקודה זו, כל העבודה שלנו התבצעה על השרת. בדפדפן השתמשנו רק כדי להציג את התוצאות.

למעשה, קשה לראות זאת אם הכל רץ על מחשב אחד. המודל בכל אופן, אמור היה לטפל בקבצים, במסגרת העניינים השוטפים במשרד ו/או בשרת האינטרנט לפני שליחת ה-HTML לדפדפן הלקוח.

בעבודה שבצד השרת, לא ניצלנו את היכולות של הדפדפן לטיפול ישיר ב-XML. אך ב-XSL Helper כן השתמשנו בחלק מן היכולות שבצד הלקוח. כאן השתמשנו באיי XML בתוך דף HTML, וניצלנו את התמיכה המובנית של הדפדפן בטעינת XML וביצוע טרנספורמציות XSL. כפי שראינו, רק IE5 תומך בXSL, וגם הוא תומך רק בטרנספורמציות, מה שבמקרה שלנו, היה מצוין.

היתרון של עיבוד בצד הלקוח, מתבטא בכך שאנו לא צריכים זכויות אדמיניסטרטיביות מיוחדות; למעשה, איננו צריכים בכלל שרת! נשים לב, כי למרות שאנו עובדים עכשיו בצד הלקוח, אנו עדיין משתמשים באותם אובייקטים ושיטות של XML כפי שהיינו משתמשים בהם בעבודה בצד השרת. הטיפול ב-XML אינו תלוי במיקומו.

החיסרון הברור, כאמור, (נכון להיום), הוא ההגבלה לסוג דפדפן אחד, Internet Explorer 5, וצריך להתמודד עם מגבלות האבטחה השונות שהן חלק מדפדפן זה. האפליקציה XSL Helper היתה דוגמה טובה לחלק מן הדברים שניתן לבצע בצד הלקוח, בסביבה מתאימה.

וזה עוד לא הכל!

אם הגעת לנקודה זו, ללא ספק הכרת את הכלים המרכזיים של XML. מזל טוב!

ובכל זאת, יש עוד!

למרות שזו טכנולוגיה חדשה יחסית, תוספות חדשות נכנסות לשוק מיום ליום בקצב מסחרר. אף ספר לא יכול לתאר את כל הדרכים האלו בפירוט, מכיון שהגלגל עדיין מסתובב, עם זאת, נוכל להמליץ לך להתפתח בכיוונים שונים בתחום זה, בעיקר לכיוונים הנכונים והמבטיחים..

פיתוח המפרט

למרות ש-XML עצמה הגיעה לשלב המלצות ב-W3C, מתרחשת עדיין עבודה רבה להרחבה ופיתוח של המפרט. ניב ה-XSL שהבאנו בספר זה היה רק בשלב טיוטה להמלצה בעת שיושם על ידי Microsoft עבור IE5, אך חלק הטרנספורמציות ב-XSL (הקרוי כעת XSLT) נדון לאחרונה כהמלצה. ישנה התקדמות גם בחלק העיצוב ב-XSL.

התאמה למפרט

בנוסף לעבודה על המפרט, ישנו מאמץ מתמשך מצד הספקים להתאמת המוצרים שלהם למפרט, במקביל להוספת הרחבות ומאפיינים המוגנים בזכויות יוצרים. מומלץ להתעדכן לעתים תדירות ב-www.w3c.org וגם באתרי הספקים השונים כדי לעקוב אחרי הטכנולוגיה המתפתחת הזו.

מאפיינים בצד הלקוח

למרות שלא העמקנו רבות ב-XSL Helper בטיפול בצד הלקוח, ישנם מאפיינים רבים אחרים בצד הלקוח המיושמים ב-IE5. ניתן לצפות ישירות ב-XML באמצעות גיליון סגנונות מובנה בברירת מחדל. ניתן לצפות ישירות ב-XML גם על ידי שימוש בהתאמה של CSS לעיצוב של מסמכים. דף ה-XML יכול להתייחס לגיליון סגנונות מפורש ב-XSL או לטפל בתוכן שלו באמצעות תסריט בצד הלקוח. בסביבה שבה ברירת הדפדפן יכולה להיות מצוינת, זה פותח אפשרויות רבות.

DTD

ספר זה השתמש ב-DTD לציון הסכימה של מסמכי ה-XML. DTDs מוכרים לכל מי שהשתמש ב-SGML, וישנם כלים שיש להם יכולת לעבוד אתם. אך עבודה עם DTDs בסביבת XML עשויה להיות מוזרה: התחביר של השפה די שונה מ-XML, לא ניתן לגשת או לשנות את המידע שב-DTD מה-DOM, ל-XSL אין כל דרך לגשת או לציין מידע ב-DTD, וחלק ניכר מהמגבלות של השפה עצמה נראה לא ברור ושרירותי. אפילו בשימושים הפשוטים ביותר ב-DTDs בספר זה, היינו חייבים לבצע עבודה רבה כדי להשיג את הפונקציונליות שרצינו. הקושי העיקרי בידי מפתחי XML הוא שהמונח XML מכיל מספר שפות שונות שיש לשנן: תחביר נדרש ל-XML בנוי היטב, XSL ו-DTD. בעוד ש-XSL דומה למפרט XML ובנוי עליו, DTD היא ממש שפה נפרדת.

החדשות הטובות הן, ששפות סכימה חדשות תופסות מקום מרכזי בשוק. הטכניקה המובילה כיום, היא טכניקת ה"סכימות" של "מיקרוסופט", שהיא מסמך XML בפני עצמו.

XML Schema

ישנן מספר הצעות וטיוטות להמלצה עבור שפת מפרט סכימטית (XML Schema), שהיא יותר טבעית ל-XML. IE5 מיישם את XML Schema, שהיא מבוססת על XML-Data ועל DCD (שניהם הערות של W3C). XML Schema נותנת את ההזדמנות לבטא מידע סכימטי באופן הרבה יותר גמיש ועקבי, אך עם הסיכון שהפרטים של המפרט ישתנו ככל שתתפתח. כאמור, המובילה את השימוש בסכימות היא "מיקרוסופט", ויש הגורסים כי טכניקה זו תתפוס יותר ויותר מקום מרכזי, כבר בעתיד הקרוב.

יכולות אחרות של שרתי SQL

אחסנו את נתוני ה-XML במסד נתונים שגרתי של SQL, בייצוג מחרוזת. לא ניצלנו ממש את היכולות הבנויות של SQL Server. למעשה, היינו יכולים לאחסן את כל קבוצת הנתונים שלנו כמסמך XML אחד גדול ולהשתמש ביכולות השאילתות של XSL (למשל,

<xsl:for-each>) או של DOM (למשל, ()selectNodes) כדי לחלץ מהקובץ רכיבים שרצינו. כמובן שאז היינו מוותרים על היתרונות של מפתוח, גיבוי, כלי מסד נתונים, וכן הלאה, אך עבור קבוצות נתונים קטנות, זו עשויה להיות גישה סבירה ללא הוצאות התקורה של מסד נתונים מלא.

אינטגרציה של XML עם מוצרים אחרים

ספקי מסדי נתונים רבים הודיעו על תוכניות לשלב יותר XML במוצרים שלהם. למשל, SQL Server 7.5 המתקרב (מכונה "Shiloh") צפוי לתמוך לחלוטין ב-XML. שפת +ASP (שם זמני) תביא שיפורים משמעותיים בעבודה עם עמודי שרת דינמיים, תתמוך ישירות ב-XML, כמו גם הגירסה הבאה של MS Visual Studio, אשר תכיל מספר רב של שירותים תומכי XML. ברור כי התפתחויות מרגשות עומדות להיות בתחום זה.

ADO 2.1

ADO 2.1 כוללת תכונות מובנות לשמירה ושחזור של קבוצות רשומות בתצורת XML. ניתן להשתמש בכך, כמו בתצורה הבינארית הקודמת המוגנת בזכויות יוצרים, לעקביות, אך מכיון שה-XML נגיש, זוהי דרך זמינה להשגת מידע מתוך קבוצות רשומות ולתוכן מאפליקציות או מערכות אחרות.

XMLHTTP

ביישום ה-XML שב-IE5 מצוי אובייקט הקרוי XMLHTTP. אובייקט זה מאפשר לשלוח ולקבל הודעות בתצורת XML בין מחשבים באמצעות HTTP. שימוש ב-HTTP להובלה מסייע בהתמודדות עם הסוגיות הרבות של מערכות הגנה (firewalls), נתבים, ותקשורת בכלל. מכיון שההודעות הן ב-XML, אנו יכולים להשתמש בכל אחת מהשיטות הרבות והולכות שבידינו לעיצוב וטיפול בתוכן.

הצעדים הבאים

למרות שאנו מעוניינים להוסיף על מה שקראנו כאן ממקורות רבים אחרים, נראה כי הכי טוב לצלול הישר לתוך Microsoft Developer Network. רשימת המידע רבת העוצמה הזו זמינה ברשת האינטרנט או על גבי תקליטור. כאן ניתן למצוא מידע הפניה נרחב כמו גם עדכונים על היישום הספציפי למיקרוסופט של המפרט.

משאב חשוב נוסף הוא כמובן האתר של World Wide Web Consortium (www.w3c.org). ישנם גם כן אתרים רבים המוקדשים לטכניקות ודיונים לגבי XML, ובכללם /http://architag.com/xmlu/, http://www.xml.com, וקבוצות הדיון ורשימות הדואר השונות.

בנוסף, לספר זה תמיכה מלאה, ובכלל זאת עדכונים, טעויות דפוס וקוד מקור, ב-

http://www.LibertyAssociates.com

פשוט צריך ללחוץ על Books&Resources.

פרק 11
Schema

בפרק זה:

- מהי XML Schema
- איך כותבים XML Schema
- בדיקת חוקיות מסמך XML מול XML Schema
- הפיכת מסמך DTD ל-XML Schema

הטכנולוגיה הנקראת XML Schema הולכת ותופסת מקום מרכזי בעבודה עם XML. פרק זה אינו קשור לרצף הפרקים שבספר, ואינו קשור לפרויקט BibelioTech, אלא מופיע כפרק העשרה.

מהי XML Schema ?

XML Schema הינה טכנולוגיה חדשה להגדרת סוג מסמך XML, אשר פותחה על ידי Microsoft על פי טיוטה של W3C.

שפת הגדרות נוספת? בשביל מה זה טוב? הרי יש לנו כבר DTD

DTD יובאה משפת ההגדרות של SGML ולכן אינה פשוטה להבנה והיא מעט מסובכת. XML Schema באה לשנות זאת, על ידי שימוש במפרט הרגיל של XML וכתיבה פשוטה, קלה וברורה.

כמו כן, XML Schema מביאה איתה מספר שינויים ותוספות שהיו חסרים ב-DTD.

להלן מספר יתרונות בולטים בשימוש ב-XML Schema:

1. XML Schema היא מסמך XML. אחת הבעיות אשר מלווה מפתחי XML, היא הצורך ללמוד את המפרט (ה-syntax) של XML, ובנוסף, את מפרט ה-DTD. למעשה, כדי לדעת XML עליך לדעת גם DTD. XML Schema מחדשת בכך שהיא בנויה מהמפרט

של XML בנוי היטב (Well formed XML), אותו אנחנו כבר מכירים. כל שעליך לדעת הוא אילו תגיות קיימות במסמך XML Schema.

2. השפה קריאה וברורה לקורא - גם התגיות נראות קלות והכתיבה מסודרת וברורה יותר לעין. השפה "מצלצלת" ברור ומבנה הסכימה נראה הגיוני.

3. אפשרויות נוספות – XML Schema מביאה אפשרויות נוספות בעת ההגדרה של רכיבים ושל תכונות רכיבים, אשר היו חסרות ב-DTD. למשל, האפשרות ליצור תכונות (Attributes) לשימוש על ידי רכיבים שונים, וכן, האפשרות לקבוע ערכי תכונות ייחודיים לרכיבים מסוימים.

4. הגדרת סוגי ערכים - יתרון משמעותי יש ל-XML Schema על DTD בכך, שניתן להגדיר בה סוגים רבים של ערכים לרכיבים ולתכונות - לא רק טקסט ו-PCDATA, כי אם ערכים מקובלים מעולם התכנות, כמו: Integer, Char, Boolean, Float וכו'.

5. תמיכה ב"מודל פתוח" Open Content Model – לעומת ה-DTD שלא איפשר להציג רכיבים ותכונות שלא הוכרזו במדויק, ב-XML Schema קיימת האפשרות של "מודל פתוח" לכתיבת רכיבים ותכונות, המאפשר הוספת אלמנטים, מעבר לאילו המוגדרים בסכימה.

6. שימוש ב-namespaces – לעומת ה-DTD, ב-XML Schema ניתן להשתמש ב-namespaces ועל ידי כך, ליצור קבוצת שמות, אשר ניתן לחזור ולהשתמש בהם. Namespaces מאפשר גם שימוש בהגדרות של סכימות חיצוניות, בזמן שב-DTD לא ניתן להשתמש בו ביותר ממסמך אחד. (את נושא ה-namespaces סקרנו בפרק 4).

מפרט XML Schema עדיין בשלבי פיתוח, אם כי, ניתן כבר להשתמש בו בצד השרת וכן, בצד הלקוח בדפדפן Internet Explorer 5 ומעלה. אני ממליץ לעקוב אחר ההתפתחות בכיוון זה, מכיון ששיטה זו מביאה איתה אפשרויות חדשות למפתחי XML, והיא תתפוס מקום מרכזי כבר בעתיד הקרוב.

לפרטים נוספים על XML Schema היכנסו לאתר Microsoft בכתובת:

http://www.msdn.microsoft.com/xml/default.asp

או לאתר W3C בכתובת:

http://www.w3.org/XML/Schema

איך כותבים XML Schema

כפי שציינתי, XML Schema היא מסמך XML לכל דבר. לכן, מסמך זה חייב להיות תואם את הנדרש ממסמך XML הבנוי היטב. בנוסף, XML Schema מורכבת מתגיות אשר מייצגות את מבנה הגדרת המסמך. תגיות אלו שייכות ל-namespace ייחודי, הן תלויות רישיות וחייבות להיכתב בדיוק כפי שאציג כאן.

כמו כל מסמך XML יש להכריז בתחילת המסמך על XML:

```
<?xml version="1.0"?>
```

רכיב השורש של המסמך הוא Schema והוא מכיל את כל יתר התגיות.

לרכיב זה תכונה בשם name המתארת את שם הסכימה ותכונה בשם xmlns המתארת את ה-namespace של הסכימה.

בדרך כלל יש לציין שני namespaces. ה-namespace הראשון של XML Schema, חייב להיות מופנה ל-urn:schemas-microsoft-com:xml-data, היא הכתובת עבור יישום ה-XML Schemaשל Microsoft.

כמו כן, אם ברצונך להשתמש בשמות סוגי נתונים, יש ליצור גם namespace עבור ה-XML Schema Data Types, באופן הבא:

```
xmlns:dt="urn:schemas-microsoft-com:datatypes"
```

להלן המבנה הכללי של מסמך XML Schema :

```
<Schema name="MySchema"
        xmlns="urn:schemas-microsoft-com:xml-data"
        xmlns:dt="urn:schemas-microsoft-com:datatypes">
.
.
.
</Schema>
```

קישור XML Schema למסמך XML

לאחר יצירת XML Schema, נרצה לקשר את מסמכי ה-XML שלנו אל הסכימה. כדי לעשות זאת, יש לשמור את מסמך ה-XML Schema ולתת לו שם בעל סיומת XML (שהרי זהו מסמך XML).

לאחר מכן, יש ליצור קישור ל-namespace ברכיב השורש של מסמך ה-XML שלנו, כמוצג בדוגמה הבאה:

```
<?xml version="1.0"?>
<MyRootElement xmlns="x-schema:http://mysite.com/MySchema.xml">
.
.
.
</MyRootElement>
```

כלומר, בתגית השורש יש להוסיף תכונה בשם xmlns אשר מייצגת namespace, אשר מקושר למסמך ה-XML Schema שלנו.

התוספת x-schema לכתובת מסמך ה-XML Schema תפקידה לציין למתרגם (ה-Parser) כי מדובר במסמך Schema על מנת שידע מה עליו לבדוק.

רכיבי XML Schema

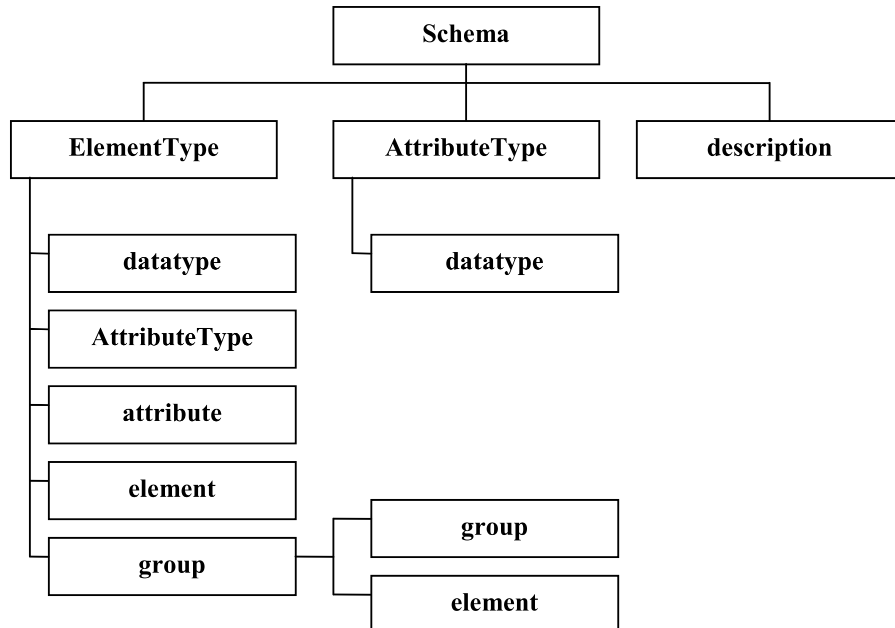
הטבלה הבאה מרכזת את הרכיבים (התגיות) של XML Schema:

שם רכיב	תיאור
<Schema>	רכיב השורש של מסמך XML Schema
<ElementType>	מגדיר סוג של רכיב במסמך XML
<AttributeType>	מגדיר סוג של תכונה במסמך XML
<element>	מופיע בתוך תחום ההכלה של התגית <ElementType> כדי להגדיר רכיב בן לרכיב המוגדר בה.
<group>	מופיע בתוך תחום ההכלה של התגית <ElementType> כדי להגדיר כיצד רכיבים בנים מקובצים בה.
<attribute>	מופיע בתוך תחום ההכלה של התגית <AttributeType> כדי להגדיר תכונה לרכיב המוגדר בה.
<datatype>	מופיע בתוך תחום ההכלה של התגית <AttributeType> או התגית <ElementType> כדי להגדיר את סוג המידע המוכל בהן.
<description>	תגית אשר מספקת מידע עבור הרכיבים: <Schema>, <ElementType> ו- <AttributeType>

רכיב השורש של כל מסמך XML Schema הוא Schema. רכיב זה מייצג את המבנה הסכימתי של מסמך ה-XML. רכיב זה יכול להכיל רכיבים בנים מסוג AttributeType, description ובעיקר רכיבי ElementType.

כעת, משהכרת את כל התגיות אשר יכולות להופיע במסמך XML Schema נסקור אותן אחת אחרי השניה, כדי ללמוד על התכונות הקיימות לכל תגית כזו.

התרשים הבא מראה את יחסי ההיררכיה האפשריים בין רכיבי XML Schema :



תרשים 1

בנוסף, כל רכיב יכול להכיל רכיב בן מסוג description.

יצירת רכיבים בעזרת ElementType

רכיב ה-ElementType של XML Schema, מייצג את רכיבי מסמך ה-XML. זהו הרכיב בו נאפיין את הרכיבים/התגיות של המסמך שלנו. למעשה, רכיב זה חופף לתגית ה-DTD <!ELEMENT>. רכיב זה יכול להכיל את הרכיבים הבנים הבאים : AttributeType, description, datatype ,element, attribute ו-group.

המבנה הכללי של רכיב זה נראה כך :

```
<ElementType
	name	=	"שם הרכיב"
	content	=	"empty" או "textOnly" או "eltOnly" או "mixed"
	model	=	"open" או "closed"
	order	=	"one" או "seq" או "many"
	dt:type	=	"סוג ערך" >
.
.
.
</ElementType>
```

כפי שניתן לראות לרכיב ה-ElementType תכונות ייחודיות אשר ערכיהן יכולים להיות שונים.

name - תכונה זו הכרחית, והיא מגדירה את שם הרכיב (התגית) במסמך ה-XML.

content - תכונה זו אופציונלית ותפקידה להגדיר את המידע המוכל ברכיב (בתוך תחום ההכלה של תגית הרכיב במסמך ה-XML).

- "empty" - מציין שהרכיב אינויכול להכיל כל סוג של מידע - טקסט או רכיבים בנים (תגית בודדת).
- "textOnly" - מציין שהרכיב אינו יכול להכיל טקסט חופשי בלבד, ואינו יכול להכיל רכיבים בנים.
- "eltOnly" - מציין שהרכיב יכול להכיל רכיבים בנים, ואינו יכול להכיל טקסט חופשי.
- "mixed" - מציין שהרכיב יכול להכיל גם רכיבים בנים וגם טקסט חופשי. זוהי ברירת המחדל.

model - תכונה זו אופציונלית והיא מביאה חידוש משמעותי לעומת DTD בכך שתפקידה הוא לקבוע האם לרכיב המסויים ניתן להכיל מידע מעבר למידע המוגדר בסכימה. מידע זה מתייחס הן לטקסט, הן לרכיבים בנים והן לתכונות של התגית, אשר אינן מוגדרות בסכימה.

להזכירכם, ב-DTD אפשרות זו אינה קיימת, וניתן ליצור אך ורק תגיות אשר הוגדרו כולן ב-DTD.

התכונה יכולה לקבל אחד מהערכים הבאים:

- "open" - ברירת המחדל המציינת כי הרכיב יכול להכיל מידע מעבר למוגדר בסכימה (טקסט, רכיבים בנים ותכונות נוספות).
- "closed" - הרכיב יכול להכיל מידע אך ורק לפי המוגדר בסכימה.

order - תכונה זו אופציונלית והיא מגדירה כמה רכיבים בנים יכול הרכיב המוגדר להכיל.

התכונה יכולה לקבל אחד מהערכים הבאים:

- "one" - הרכיב יכול להכיל רק רכיב אחד בן אחד מתוך רשימת הרכיבים האפשריים עבורו ולא יותר.
- "seq" - הרכיב חייב להכיל את כל הרכיבים הבנים המוגדרים עבורו, על פי סדר הגדרתם.
- "many" - ברירת המחדל המגדירה כי הרכיב יכול שלא להכיל רכיבים בנים, להכיל רכיב אחד, כמה רכיבים, או את כל הרכיבים על פי סדר כלשהו.

dt:type - תכונה זו מגדירה את סוג המידע של הטקסט המוכל ברכיב. אחד ההבדלים המשמעותיים בין XML Schema ו-DTD, הוא השימוש בסוגי מידע. עבור XML Schema הוגדרו סוגי מידע ייחודיים, חלקם קיימים בשפות תכנות מקובלות כמו VB, C++, C, Java וחלקם חדשים.

להזכירכם, בראש קובץ הגדרות ה-XML Schema אנו מפנים ל-namespace של הגדרות סוג מידע אלו. ללא הפניה זו, לא תוכל להשתמש בהגדרות הסוגים.

את רשימת הערכים האפשריים תמצא בטבלה שבסוף הפרק. בין הערכים הקיימים, תמצא את "time", "date", "char", "boolean", "number", "int", "float", "string" ועוד.

הגדרת רכיבים בנים בעזרת element

כדי להגדיר ב-XML Schema היררכיה של אב-בן, יש להשתמש ברכיב element.

את תגיות element יש למקם בתוך תחום ההכלה של ElementType או של group (הסבר בהמשך). הרכיב element מייצג מופע של אחד הרכיבים שהגדרנו במקום אחר במסמך על ידי רכיב ElementType.

התגית element הינה תגית בודדת. והמבנה הכללי שלה נראה כך :

```
<element
       type          =         "שם הרכיב"
       minOccurs  =         "1" או "0"
       maxOccurs  =         "1" או "*" />
```



שים לב! קיים הבדל משמעותי בין: ElementType **ו-** element type.
ElementType **- היא תגית ליצירת רכיב חדש.**
element type **- הוא רכיב** element **ויש לו תכונה בשם** type.

כפי שניתן לראות, לרכיב element שלוש תכונות ומשמעותן :

type - תכונה זו היא הכרחית, והיא מגדירה את שם הרכיב הבן, כפי שהוגדר קודם לכן.

minOccurs - תכונה זו אופציונלית, והיא מגדירה את מספר המופעים המינימלי של רכיב זה, בתוך רכיב האב.

התכונה יכולה לקבל אחד מהערכים הבאים :

- "1" - ברירת המחדל המגדירה כי הרכיב חייב להופיע לפחות פעם אחת בתחום ההכלה של רכיב האב.
- "0" - הרכיב אופציונלי ויכול גם שלא להופיע בתחום ההכלה של רכיב האב.

maxOccurs - תכונה זו היא אופציונלית, והיא מגדירה את מספר המופעים המקסימלי של רכיב זה, בתוך רכיב האב.

- "1" - ברירת המחדל המגדירה כי הרכיב חייב להופיע לכל היותר פעם אחת בתחום ההכלה של רכיב האב.
- "*" - אין הגבלה למספר המופעים של הרכיב בתחום ההכלה של רכיב האב.

אופן השימוש ברכיבים נעשה בשלבים הבאים :

1. אנחנו מגדירים את כל סוגי הרכיבים הקיימים באמצעות ElementType, באופן מרוכז, על פי סדר ההיררכיה, בתחילת מסמך ה-XML Schema.

2. בתוך תחום ההכלה של רכיבים אבות, או קבוצות רכיבים, אנחנו מציבים רכיבי element עבור כל רכיב בן. ייתכן כמובן שרכיב יופיע מספר פעמים כרכיב בן של רכיבים שונים, לכן את הגדרתו נעשה על ידי ElementType, ואת סידורו ההיררכי כרכיב בן של רכיב אחר, נגדיר על ידי element בתוך תחום ההכלה של הגדרת רכיב האב, על ידי ElementType.

לדוגמה, הבה נאמר כי מעוניינים להגדיר ארבעה רכיבים: רכיב name, רכיב gender, רכיב manager, ורכיב worker. ארבעת הרכיבים מייצגים שם, מין, מנהל ועובד בהתאמה. רכיב name ורכיב gender הינם רכיבים בנים הן של manager ושל worker, ולכן יש להכילם בשניהם.

נגדיר את הרכיבים הקיימים. ראשית, רכיבי name ו-gender, מכיוון שאלו רכיבים אשר מכילים אך ורק טקסט, ואינם אבות לרכיבים נוספים, נשתמש בכתיבה של תגית בודדת ב-XML (סימן לוכסן בסוף תגית):

```
< ElementType name="name" content="textOnly" dt:type="string"/>
< ElementType name="gender" content="textOnly" dt:type="string"/>
```

כעת נגדיר את שני הרכיבים הנותרים, אשר מכילים רכיב name ורכיב gender כל אחד. שים לב שמבנה התגית ElementType כאן הוא כמו תגית הכלה, ואילו תגיות ה-element הינן תגיות בודדות:

```
<ElementType name="manager" content="eltOnly" order="seq">
      <element type="name" minOccurs="1" maxOccurs="1" />
      <element type="gender" minOccurs="1" maxOccurs="1" />
</ElementType>

<ElementType name="worker" content="eltOnly" order="seq">
      <element type="name" minOccurs="1" maxOccurs="1"/>
      <element type="gender" minOccurs="1" maxOccurs="1"/>
</ElementType>
```

כמובן שכל תגית יכולה להכיל תגית אחרת, כך שאם היה לנו רכיב נוסף בשם "employee" אשר מייצג עובד, וחייב להכיל רכיב בן אחד, מסוג manager או worker אזי ההכרזה עליו היתה נראית כך:

```
<ElementType name="employee" content="eltOnly" order="one">
      <element type="manager" minOccurs="1" maxOccurs="1"/>
      <element type="worker" minOccurs="1" maxOccurs="1"/>
</ElementType>
```

הגדרת הרכיב האחרון בסכימה תהיה הגדרת ElementType של רכיב השורש של מסמך ה-XML. הדוגמה הבאה יוצרת רכיב שורש בשם personal אשר יכול להכיל מספר רכיבים בנים מסוג employee:

```
<ElementType name="personal" content="eltOnly" order="seq">
      <element type="employee" minOccurs="0" maxOccurs="*"/>
</ElementType>
```

יצירת קבוצות רכיבים בעזרת group

הרכיב group מאפשר יצירה של קבוצות רכיבים. הרעיון הוא ליצור קבוצה המורכבת ממספר רכיבים לפי סדר, וכפופה לרכיב אב אשר מוגדר בעזרת ElementType. רכיב הקבוצה הינו רכיב בן של ElementType ולכן התגית עצמה תיכתב בתוך תחום ההכלה של ElementType.

המבנה הכללי של תגית ה-group נראה כך:

```
<group
        minOccurs          =          "0" או "1"
        maxOccurs          =          "1" או "*"
        order              =          "one" או "seq" או "many">
.
.
.
</group>
```

התגית תיכתב בתוך ה-ElementType של רכיב האב של הקבוצה, ובתוך תחום ההכלה שלה יסודרו רכיבי הקבוצה בזה אחר זה על ידי הכרזות element.

את שמות התכונות של רכיב group כבר הכרת קודם, הבה נבין את משמעותן עבור הקבוצה:

minOccurs - תכונה זו אופציונלית, והיא מגדירה את מספר המופעים המינימלי של רכיב זה, בתוך רכיב האב.

התכונה יכולה לקבל אחד מהערכים הבאים:

- "1" - ברירת המחדל המגדירה כי הקבוצה חייבת להופיע לפחות פעם אחת בתחום ההכלה של רכיב האב.
- "0" - הקבוצה אופציונלית ויכולה גם לא להופיע בתחום ההכלה של רכיב האב.

maxOccurs - תכונה זו אופציונלית, והיא מגדירה את מספר המופעים המקסימלי של רכיב זה, בתוך רכיב האב.

- "1" - ברירת המחדל המגדירה כי הקבוצה חייבת להופיע לכל היותר פעם אחת בתחום ההכלה של רכיב האב.
- "*" - אין הגבלה למספר המופעים של הקבוצה בתחום ההכלה של רכיב האב.

order - תכונה זו אופציונלית והיא מגדירה כמה רכיבים בנים יכולה הקבוצה המוגדרת להכיל.

התכונה יכולה לקבל אחד מהערכים הבאים:

- "one" - הקבוצה יכולה להכיל רק רכיב אחד בן אחד מתוך רשימת הרכיבים האפשריים עבורה ולא יותר.

❖ "seq" - הקבוצה חייבת להכיל את כל הרכיבים הבנים המוגדרים עבורה, לפי סדר הגדרתם.

❖ "many" - ברירת המחדל המגדירה כי הקבוצה יכולה שלא להכיל רכיבים בנים, להכיל רכיב אחד, כמה רכיבים, או את כל הרכיבים לפי סדר כלשהו.

בדוגמה הבאה הגדרתי רכיב download המייצג, לצורך העניין, פרטי היפר-קישור להורדת קובץ מאתר ברשת. הרכיב מכיל רכיב בן בשם title המייצג את שם הקובץ, רכיב abstract למלל חופשי, ואחר כך קבוצה המתארת את פרטי הקובץ ומכילה את הרכיבים license, date, format, size בזה אחר זה:

```
<ElementType name="download " content="eltOnly" order="seq">
        <element type="title" />
        <element type="abstract" />
        <group order="one">
              <element type="size" />
              <element type="format" />
              <element type="date" />
              <element type="license" />
        </group>
</ElementType>
```

כמובן, שכל רכיב element שצוין כאן, הוגדר כבר קודם לכן בעזרת ElementType משלו.

יצירת תכונות על ידי AttributeType

תפקידו של הרכיב AttributeType הוא ליצור תכונות, אשר ניתן יהיה להשתמש בהן, במסמך XML.

רכיב זה רק יוצר את התכונות האפשריות, הוא אינו משייך אותן לרכיבי XML. את השיוך נעשה מאוחר יותר בסכימה, על ידי הרכיב attribute.

זה דומה לאופן השימוש ב-ElementType ו-element. כפי שאנו מגדירים רכיבים על ידי ElementType ומשייכים אותם לרכיבי אב בעזרת element כך, אנו ראשית מגדירים את התכונות האפשריות על ידי הגדרת כל תכונה בנפרד בעזרת AttributeType ואחר כך משייכים לכל רכיב XML שהגדרנו, את התכונות שלו, על ידי רכיבי element.

הרעיון הוא לאפשר ליצור תכונות, אשר ניתן להשתמש בהן ברכיבים שונים. הדבר דומה לתכונות height ו-width של HTML, אשר יכולות לשמש גם את התגיות IMG, TABLE, EMBED ותגיות נוספות אחרות.

תגית הרכיב הינה תגית בודדת והמבנה הכללי שלה נראה כך:

```
<AttributeType
        name            =       "שם התכונה"
        required        =       "yes" או "no"
        dt:type         =       "סוג מידע"
        dt:values       =       "רשימת ערכים מוגדרים מראש"
        default         =       "ערך ברירת מחדל"  />
```

תגית הרכיב יכולה לקבל את התכונות הבאות:

name – תכונה זו היא הכרחית, והיא קובעת את שם התכונה המוגדרת.

required – תכונה זו קובעת האם חובה לתת ערך לתכונה המוגדרת. לתכונה שני ערכים אפשריים:

"yes" – חובה לתת ערך לתכונה.

"no" – אין חובה לתת ערך לתכונה.

dt:type – תכונה זו מגדירה את סוג המידע של ערכי התכונה. הסוגים המותרים הם הסוגים שהוגדרו ל-XML Schema (ראה בטבלאות שבסוף הספר). כמו כן, תכונה זו יכולה לקבל את הערך "enumeration".

dt:values - כאשר ערך התכונה dt:type הוא "enumeration", ניתן ליצור ערכים מוגדרים מראש, ולהגדיר אותם ברשימה בתכונה זו. על הערכים להיות מופרדים זה מזה על ידי תו רווח.

default – תכונה זו קובעת את ערך ברירת המחדל של התכונה המוגדרת.

למשל, בדוגמה הבאה יצרתי תכונה בשם day אשר יכולה לקבל אך ורק את הערכים Sun, Mon, Tue, Wed, Thu, Fri, Sat . קבעתי שזוהי תכונה הכרחית, וערך ברירת המחדל שלה הינו "Sun":

```
<AttributeType
	name		=	"day"
	required	=	"yes"
	dt:type		=	"enumeration"
	dt:values	=	"Sun Mon Tue Wed Thu Fri Sat"
	default		=	"Sun"	/>
```

שיוך תכונות לרכיבים בעזרת attribute

הרכיב attribute הינו רכיב בן של ElementType ותפקידו להגדיר מהן התכונות האפשריות, (אשר הוגדרו קודם לכן על ידי AttributeType), עבור הרכיב המוגדר.

התגית של רכיב זה היא תגית בודדת, ובתוכה, בעזרת תכונות משלה, מוגדרת תכונת רכיב ה-XML הנדרש.

תגית הרכיב attribute הינה תגית בודדת והמבנה הכללי שלה נראה כך:

```
<attribute
	type		=	"סוג מידע"
	required	=	"yes" או "no"
	default		=	"ערך ברירת מחדל"  />
```

התגית יכולה לקבל את התכונות הבאות:

type – תכונה זו הכרחית והיא מציינת את שם התכונה, כפי שהוגדרה קודם לכן על ידי AttributeType.

required – מציין האם התכונה המוגדרת חייבת לקבל ערך על ידי הרכיב המוגדר. לתכונה זו שני ערכים אפשריים:

"yes" – התכונה המוגדרת חייבת לקבל ערך על ידי הרכיב המוגדר.
"no" – התכונה אינה חייבת לקבל ערך על ידי הרכיב המוגדר.

default – תכונה זו קובעת את ערך ברירת המחדל עבור התכונה המוגדרת, במסגרת הרכיב המסוים המוגדר. ערך זה ידרוס את הערך שקיבלה התכונה קודם לכן, בהגדרת ה-AttributeType שלה.

בדוגמה הבאה, יצרתי רכיב בשם car בעל התכונות: color, אשר יכולה לקבל את הערכים "red","blue","green" ו-"yellow", תכונה בשם system אשר יכולה לקבל את הערכים "Auto" או "Manual", תכונה בשם height אשר מקבלת ערך עשרוני ותכונה בשם year המקבלת תאריך. הרכיב car שיצרתי כאן, אינו מכיל רכיבים בנים (תגית בודדת):

```
<AttributeType name="color" dt:type="enumeration" dt:values="red blue green yellow"
/>
<AttributeType name="system" dt:type="enumeration" dt:values="auto manual" />
<AttributeType name="height" dt:type="float" />
<AttributeType name="year" dt:type="date" />

<ElementType name="car" content="empty" >
        <attribute type="color" default="blue" required="yes" />
        <attribute type="system" default="auto" required="yes" />
        <attribute type="height" required="yes" />
        <attribute type="year" required="yes" />
</ElementType>
```

כאמור, יצרתי כאן רכיב car אשר אין לו רכיבים בנים. אם בכל זאת הייתי בוחר להוסיף גם רכיבים בנים, הייתי מכריז עליהם לפני כן בעזרת ElementType מתאים, ומוסיף מופעים שלהם אל תוך תחום ההכלה של תגית הרכיב car בעזרת element.



שים לב! בחרתי לקבוע את ערך required של התכונות בתוך תגיות ה-attribute ולא בתגית ההגדרה של התכונות AttributeType. אפשרות זו היא דוגמה מצוינת לניידות של XML Schema. היא מאפשרת לקבוע עבור אילו רכיבים תכונה מסוימת היא הכרחית, ומותירה את האפשרות ליצור רכיבים נוספים עם התכונה הזו, שעבורם היא אינה הכרחית.

כך, למשל, עבור הרכיב car התכונה height הוגדרה כתכונה לא הכרחית. בהמשך הסכימה יהיה ניתן ליצור רכיב אחר, שעבורו תכונה זו כן הכרחית.

קביעת סוג מידע בעזרת datatype

רכיב datatype של XML Schema, יכול להיות רכיב בן של ElementType או AttributeType.

תפקידו להגדיר את סוג המידע המוכל ברכיב או התכונה. לרכיב זה קיימת רק התכונה dt:type אשר ערכה מציין את סוג המידע.

ניתן להשתמש בתגית הרכיב כתגית בודדת בתוך תחום ההכלה של תגיות הרכיבים ElementType או AttributeType, באופן הבא:

```
<datatype dt:type="char" />
```

למרות זאת, ראינו כבר כי הן ל-ElementType והן ל-AttributeType קיימת התכונה dt:type, אשר מגדירה את סוג המידע. לכן, השימוש בתגית הרכיב datatype אינו נהוג.

רכיב התיאור description

רכיב ה-XML Schema האחרון שנותר להכיר נקרא description.

רכיב זה משמש לתיאור של רכיבים, ואינו עובר עיבוד. ניתן להשתמש בתגית הרכיב בכל מקום בו תרצו להוסיף הערות, הסברים או תיאור של אלמנטים שהגדרתם.

תגית הרכיב הינה תגית מכולה, ובתחום הכלתה יהיה טקסט חופשי:

```
<description>
I don't know why I added this description Tag here...
I guess I'm stuck with my Schema...
</description>
```

בדיקת חוקיות מסמך XML מול XML Schema

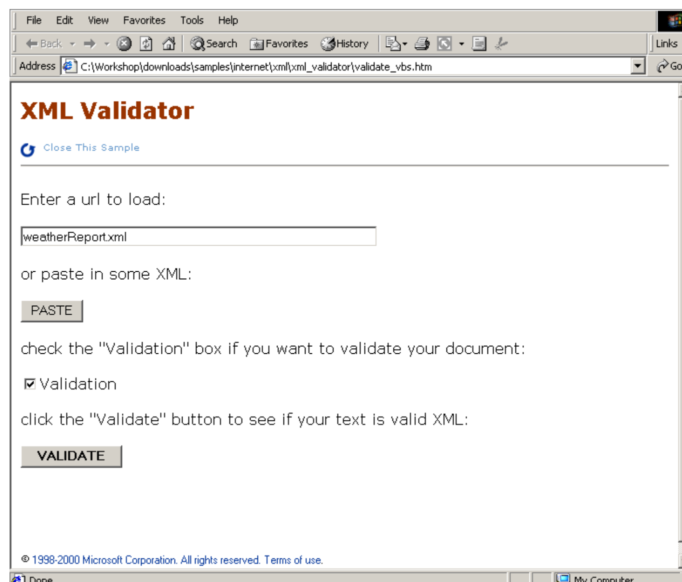
לאחר שיצרנו מסמך XML Schema ומסמך XML, וקישרנו בין המסמכים באמצעות הקישור בתגית רכיב השורש של מסמך ה-XML, היינו רוצים לבדוק את חוקיות המסמך.

ודאי תיארת לעצמך, כי די בטעינת מסמך ה-XML כפי שהוא לדפדפן כדי שזה יבדוק את חוקיותו.

אך לצערנו, לא כך הדבר. כל שעושה הדפדפן במקרה זה, הוא רק בדיקה האם המסמך והסכימה בנויים היטב ותו לא. כדי לבדוק חוקיות של מסמך XML אל מול הסכימה, יש צורך בכתיבת סקריפט מתאים לכך.

ניתן גם, מצד שני, להשתמש בתוכנית עזר לבדיקת חוקיות. תוכנית כזו, בשם XML Validator, מופצת באופן חופשי על ידי Microsoft וניתנת להורדה מאתר MSDN בכתובת הבאה:

http://msdn.microsoft.com/downloads/samples/internet/xml/xml_validator



תרשים 2

כפי שניתן לראות, זוהי תוכנית הבנויה מסקריפט ורצה על גבי הדפדפן.

התוכנית מגיעה בשתי גרסאות, גירסה הכתובה בשפת JScript וגירסה הכתובה בשפת VBScript.

כמו כן, בתיקיה בה תתקין את התוכנה תמצא שני קבצי XML:

1. WeatherReport.xml – מסמך XML המתאר דיווח מזג אויר.
2. WeatherSchema.xml – מסמך סכימה המתאים ל-XML.

התוכנית בודקת שני דברים:

1. האם המסמך בנוי היטב (האם הוא Well Formed).
2. האם המסמך חוקי (מתאים להגדרות הסכימה).

כל שנדרש כדי לבדוק מסמך XML הוא להכניס את מיקום הקובץ בתיבת הטקסט "Enter a url to load" ולאחר מכן, ללחוץ על מקש "VALIDATE" שבתחתית העמוד.

הבה ונביט בקבצי הדוגמה אשר מגיעים עם התוכנית.

להלן תוכן הקובץ WeatherReport.xml :

```
<?xml version="1.0"?>
<WEATHERREPORT xmlns="x-schema:WeatherSchema.xml">
  <STATE NAME="California">
    <CITY NAME="Los Angeles">
      <SKIES VALUE="PARTLYSUNNY"/>
      <HI C="31" F="87"/>
      <LOW C="18" F="65"/>
```

```
      Partly cloudy
    </CITY>
    <CITY NAME="Sacramento">
      <SKIES VALUE="SUNNY"/>
      <HI C="36" F="97"/>
      <LOW C="17" F="64"/>
      Sunny and hot.
    </CITY>
    <CITY NAME="San Diego">
      <SKIES VALUE="PARTLYSUNNY"/>
      <HI C="26" F="78"/>
      <LOW C="19" F="67"/>
    </CITY>
    <CITY NAME="San Fransisco">
      <SKIES VALUE="PARTLYCLOUDY"/>
      <HI C="26" F="79"/>
      <LOW C="14" F="58"/>
      Partly cloudy and humid
    </CITY>
    <CITY NAME="Truckee">
      <SKIES VALUE="RAIN"/>
      <HI C="32" F="89"/>
      <LOW C="11" F="52"/>
      Scattered thunderstorms
    </CITY>
    </STATE>
  <STATE NAME="New Jersey">
    <CITY NAME="Newark">
      <SKIES VALUE="PARTLYSUNNY"/>
      <HI C="36" F="97"/>
      <LOW C="21" F="71"/>
      Partly sunny, breezy and humid
    </CITY>
    <CITY NAME="Trenton">
      <SKIES VALUE="PARTLYCLOUDY"/>
      <HI C="32" F="90"/>
      <LOW C="18" F="65"/>
      Partly cloudy and humid
    </CITY>
    <CITY NAME="Princeton">
      <SKIES VALUE="RAIN"/>
      <HI C="33" F="92"/>
      <LOW C="20" F="68"/>
      Thundershowers
    </CITY>
```

```
    <CITY NAME="White Meadow Lake">
      <SKIES VALUE="SUNNY"/>
      <HI C="24" F="85"/>
      <LOW C="21" F="70"/>
      Sunny, clear skies, breezy and warm.
    </CITY>
  </STATE>
  <STATE NAME="Washington">
    <CITY NAME="Seattle">
      <SKIES VALUE="RAIN"/>
      <HI C="20" F="68"/>
      <LOW C="15" F="59"/>
      Raining on and off throughout the day.
    </CITY>
    <CITY NAME="Yakima">
      <SKIES VALUE="PARTLYSUNNY"/>
      <HI C="23" F="73"/>
      <LOW C="14" F="57"/>
      Partly sunny after morning clouds
    </CITY>
    <CITY NAME="Redmond">
      <SKIES VALUE="SNOW"/>
      <HI C="2" F="35"/>
      <LOW C="-7" F="20"/>
      Snowstorms in the afternoon
    </CITY>
  </STATE>
</WEATHERREPORT>
```

תוכן מסמך הסכימה המתאים נראה כך :

```
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
    xmlns:dt="urn:schemas-microsoft-com:datatypes">

 <AttributeType name="C" required="yes" dt:type="string"/>

<AttributeType name="F" required="yes" dt:type="string"/>

 <ElementType name="LOW">
  <attribute type="C"/>
  <attribute type="F"/>
 </ElementType>

 <ElementType name="HI">
  <attribute type="C"/>
  <attribute type="F"/>
 </ElementType>
```

```
  <AttributeType name="VALUE" required="yes" dt:type="enumeration"
    dt:values="SUNNY PARTLYSUNNY PARTLYCLOUDY CLOUDY RAIN SNOW"/>

  <ElementType name="SKIES">
    <attribute type="VALUE"/>
  </ElementType>

  <AttributeType name="NAME" required="yes" dt:type="string"/>

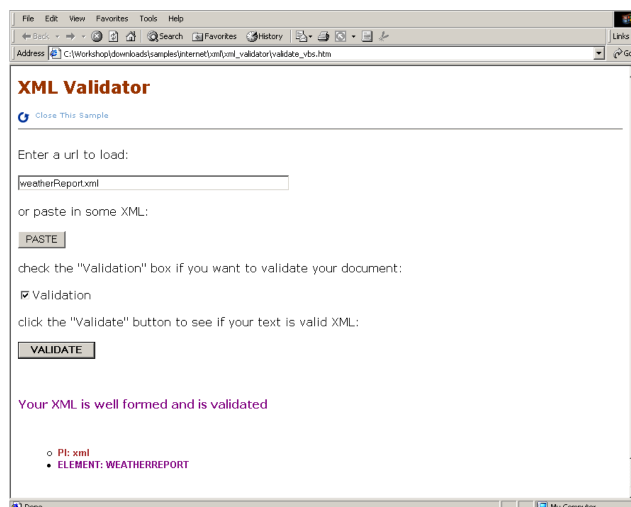
  <ElementType name="CITY" content="mixed">
    <attribute type="NAME"/>
    <element type="SKIES"/>
    <element type="HI"/>
    <element type="LOW"/>
  </ElementType>

  <ElementType name="STATE" content="mixed">
    <attribute type="NAME"/>
    <element type="CITY"/>
  </ElementType>

  <ElementType name="WEATHERREPORT" content="eltOnly">
    <element type="STATE"/>
  </ElementType>

</Schema>
```

קובץ XML זה הינו חוקי ועומד בהגדרות הסכימה. בבדיקת חוקיות הקובץ בתוכנית התקבלו התוצאות הבאות:

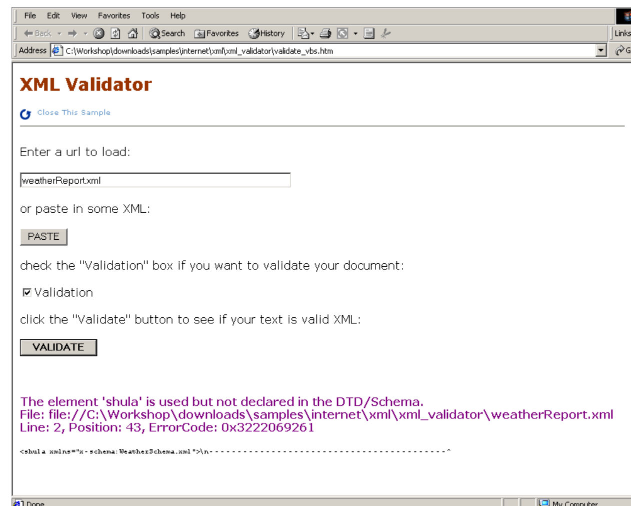


תרשים 3

כעת, הבה נשתעשע עם התוכנית, ונשנה את שם תגית השורש הפותחת ל-shula באופן הבא:

```
<?xml version="1.0"?>
<shula xmlns="x-schema:WeatherSchema.xml">
...
```

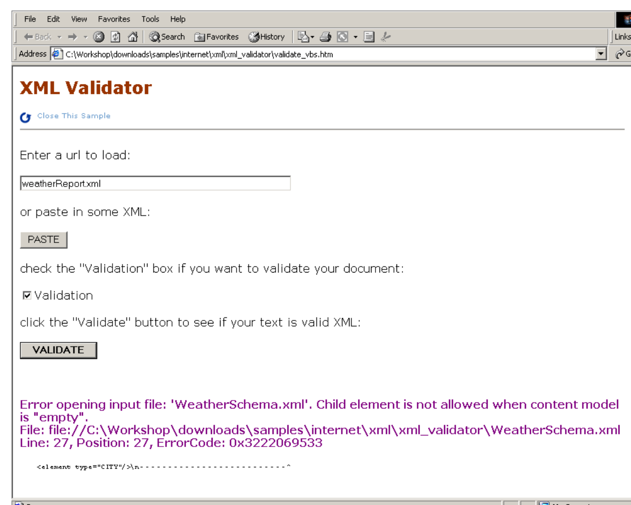
נריץ ונקבל:



תרשים 4

קיבלנו הודעת שגיאה, אשר מציינת כי המסמך אינו חוקי. הרכיב shula אינו מופיע בהגדרות הסכימה. באותו אופן נקבל הודעת שגיאה עבור כל הצגה בלתי חוקית של המסמך.

ההודעה הבאה התקבלה אחרי שהתכונה content של הרכיב state שונתה מ-"many" ל-"empty":



תרשים 5

הפיכת מסמך DTD ל-XML Schema

אחד הדברים שמשתמשי DTD מבקשים לעשות כשהם מגלים את XML Schema, הוא להעביר את מסמכי ה-DTD שלהם לסכימה. ניתן לעשות זאת בשתי דרכים:

1. בעבודה קשה – רכיב, רכיב.
2. בעזרת תוכנית מתרגמת.

המרת מסמך DTD ל-XML Schema בדרך הקשה

אמנם, כשיש תוכניות עזר, אין צורך "ללכלך את הידיים". ובכל זאת, כדאי להבין את המעבר בין DTD ל-XML Schema, מכיוון שהבנה זו תעזור לך להבין את אופיים של רכיבי XML Schema טוב יותר.

להלן הצעדים לתרגום מסמך DTD ל-XML Schema.

1. ראשית, יש ליצור את כותרת מסמך XML Schema.
2. אחר כך נעבור על מסמך DTD מלמעלה למטה. כלומר, מהרכיב העליון לתחתון. שים לב שאחד ההבדלים בין DTD ו-XML Schema הוא אופן סידור הרכיבים במסמך. ב-DTD הרכיבים מסודרים מלמעלה למטה לפי סדר חשיבותם, וב-XML Schema תמצא קודם את ההצהרות של כל רכיב ותכונה ובסוף המסמך ימוקמו הרכיבים לפי סדר היררכיה מהנמוכים ביותר אל רכיב השורש. לכן, עלינו להעתיק את הרכיבים שבתחילת ה-DTD לתחתית מסמך ה-XML Schema.

אין די בהעתקת הרכיבים מתגיות ה- <!ELEMENT> של DTD אל תוך רכיבי AttributeType של XML Schema, מכיוון שקיימות גם הגדרות לגבי אופן הופעת הרכיבים במסמך XML, מספר המופעים המותר, סדר הרכיבים, קיבוץ וכו' לכן, יש צורך גם לתרגם את אופי ההגדרות הנ"ל מצורת הכתיבה ב-DTD ל-XML Schema.

הטבלה הבאה מרכזת חלק חשוב מהכפילויות של הגדרות DTD עבור XML Schema:

צורת הכתיבה ב-DTD	צורת הכתיבה ב-XML Schema
שימוש בסוגריים לקיבוץ רכיבים: <!ELEMENT book (section)>	סדר את הרכיבים על ידי רכיבי element בתוך תגית הרכיב ElementType או group: <ElementType name="book"> <element type="section" /> </ElementType>
שימוש בפסיק לקביעת סדר הופעת הרכיבים בזה אחר זה: <!ELEMENT book (title, author)>	קבע את ערך התכונה order של הרכיבים AttributeType או group ל-"seq": <ElementType name="book" order="seq"> <element type="title" /> <element type="author" /> </ElementType>

<table>
<tr><th>צורת הכתיבה ב-DTD</th><th>צורת הכתיבה ב-XML Schema</th></tr>
<tr><td>שימוש ב-pipe לבחירה של רכיבים :
<!ELEMENT title (shortTitle | longTitle)></td><td>קבע את ערך התכונה order של הרכיבים AttributeType או group ל-"many" :
<ElementType name="title" order=" many">
<element type="shortTitle" />
<element type="longTitle" />
</ElementType></td></tr>
<tr><td>שימוש ב-pipe לבחירה של ערכי תכונה :
<!ATTLIST color
type (blue | red | green) blue ></td><td>שנה את ערך התכונה dt:type של הרכיב AttributeType ל-"enumeration" וקבע רשימת ערכים מוגדרים מראש ב-dt:values :
<AttributeType
name="color"
dt:type="enumeration"
dt:values="blue red green"
default="blue" /></td></tr>
<tr><td>שם רכיב בן בתגית <!ELEMENT> ללא אופרטור :
<!ELEMENT book (section)></td><td>קבע את ערכי התכונה minOccurs של element או group ל-"1", ואת ערכי התכונה maxOccurs ל-"1" :
<ElementType book>
<element section minOccurs="1"
maxOccurs="1" />
</ElementType></td></tr>
<tr><td>שם רכיב בן עם הסימן ? :
<!ELEMENT book (section?) ></td><td>קבע את ערכי התכונה minOccurs של element או group ל-"0", ואת ערכי התכונה maxOccurs ל-"1" :
<ElementType book>
<element section minOccurs="0"
maxOccurs="1" />
</ElementType></td></tr>
<tr><td>שם רכיב בן עם הסימן + :
<!ELEMENT book (section+) ></td><td>קבע את ערכי התכונה minOccurs של element או group ל-"1", ואת ערכי התכונה maxOccurs ל-"*" :
<ElementType book>
<element section minOccurs="1"
maxOccurs="*" />
</ElementType></td></tr>
</table>

צורת הכתיבה ב-DTD	צורת הכתיבה ב-XML Schema
שם רכיב בן עם הסימן * : <!ELEMENT book (section*) >	קבע את ערכי התכונה minOccurs של element או group ל-"0", ואת ערכי התכונה maxOccurs ל-"*": <ElementType book> <element section minOccurs="0" maxOccurs="*" /> </ElementType>

ודאי הבנת את הרעיון הכללי. למעשה, ניתן לתרגם כל מסמך מ-DTD ל-XML Schema די בקלות. עם זאת, הדרך ההפוכה יכולה להיות ארוכה הרבה יותר, מה גם שאנחנו לא ממש רוצים לעשות את זה.

בדוגמה הבאה נציג מסמך DTD פשוט, ומיד לאחר מכן, נמיר אותו ל-XML Schema:

```
<!ELEMENT class (name , teacher+ , student*, assistant?)>
<!ATTLIST
        name CDATA #IMPLIED
        type (Math | History | Drama) "Math">

<!ELEMENT teacher (#PCDATA)>
<!ELEMENT student (#PCDATA)>
<!ELEMENT assistant (#PCDATA)>
```

כעת ניגש למלאכת התרגום ל-XML Schema.

ראשית, נכתוב את הצהרת XML ורכיב Schema:

```
<?xml version="1.0"?>
<Schema name="ClassSchema"
        xmlns="urn:schemas-microsoft-com:xml-data"
        xmlns="urn:schemas-microsoft-com:datatypes">
.
.
.
</Schema>
```

המלצה! ישנם כותבי HTML ו-XML רבים אשר כותבים את תגיות המסמך לפי סדר הופעתן במסמך, כך שכל כמה רגעים הם עוצרים לחפש האם ישנה איזושהי תגית מכולה ששכחו לסגור. לעיתים קרובות, הם גם שוכחים לסגור אותן. ב-HTML הסלחנית "מחדלים" שכאלו עוברים מבלי לשים לב. ב-XML הקשיחה זה לא יעבור בשקט. לכן, אני ממליץ לסגור כל תגית מכולה כבר כשאתם פותחים אותה. אחר כך תמלאו את תחום ההכלה שלה בהתאם. כך תחסכו לעצמכם הרבה כאב ראש מיותר. אגב, באותה הדרך אני נוהג בתכנות – ב-C, או Java, למשל, אני מייד סוגר כל אזור תחום בסוגריים מסולסלים, כמו פונקציות או משפטי IF. הקומפיילר שלי אף פעם לא צועק עלי בעניין הזה.

לאחר יצירת רכיב ה-Schema, ניצור את יתר הרכיבים.

כאמור, אנחנו עוברים על מסמך DTD מלמעלה למטה. לכן, עלינו ליצור רכיב בשם class. מכיוון שרכיב זה מכיל רכיבים אשר ב-XML Schema אמורים להיות מוגדרים קודם לכן, נכתוב את תגית הרכיב ElementType בסוף קובץ ה-XML Schema ואחר כך, נמלא את האזור שלפניו בהגדרות שאר הרכיבים.

```
<ElementType name="class" content="eltOnly" order="seq">
	<element type="teacher" minOccurs="1" maxOccurs="*" />
	<element type="student" minOccurs="0" maxOccurs="*" />
	<element type="assistant" minOccurs="0" maxOccurs="1" />
	<attribute type="name" />
	<attribute type="type" />
</ElementType>
```

בבניית הרכיב הנ"ל השתמשתי בטבלה שבעמוד הקודם.

כעת כל שנותר הוא ליצור את רכיבי ElementType של כל יתר הרכיבים ואת רכיבי AttributeType של התכונות. להלן הסכימה המלאה:

```
<?xml version="1.0"?>
<Schema name="ClassSchema"
	xmlns="urn:schemas-microsoft-com:xml-data"
	xmlns="urn:schemas-microsoft-com:datatypes">

<ElementType name="teacher" content="textOnly" / >

<ElementType name="student" content="textOnly" / >

<ElementType name="assistant" content="textOnly" / >

<AttributeType
name="type"
dt:type="enumeration"
dt:values="Math History Drama"
default="Math"/ >

	<AttributeType name="name" dt:type="string" />

<ElementType name="class" content="eltOnly" order="seq">
		<element type="teacher" minOccurs="1" maxOccurs="*" />
	<element type="student" minOccurs="0" maxOccurs="*" />
		<element type="assistant" minOccurs="0" maxOccurs="1" />
	<attribute type="name" />
	<attribute type="type" />
</ElementType>

</Schema>
```

המרת מסמך DTD ל-XML Schema בעזרת תוכנית מתרגמת

אחרי שהבנו כיצד פועלת ההמרה מ-DTD ל-XML Schema, אספר לך שקיימות תוכניות המרה אשר פותרות אותך מעבודה זו. תוכנית מתרגמת היא פתרון מהיר ונוח, אם כי לעיתים רחוקות אתה עלול להיתקל ב"נפילות" בלתי צפויות, אך זה אמור בעיקר לגבי מסמכי DTD מסובכים למדי.

מכאן לאן?

על פי המגמות המסתמנות כיום, XML Schema תלך ותתפוס תפקיד מרכזי בהגדרת מסמכי XML.

את ההתפתחות של XML Schema מובילה כיום Microsoft, אם כי הסכימה על פי Microsoft שונתה מאוד מאז פורסמה הטיוטה של W3C בנושא. גם W3C מפתחת במקביל את XML Schema שלה. נכון לכתיבת שורות אלו, עדיין לא פורסמה המלצה רשמית של האיגוד, ונראה כי הדרך עוד רבה לפרסום ההמלצה הרשמית. בכל מקרה, ברור הוא כי Microsoft עושה הרבה לפיתוח הסכימה, והפיכתה לעובדה קיימת. כבר היום יש שימוש נרחב ב-XML Schema ונראה כי הפשטות, הניידות, והיכולות החדשות שהיא מביאה איתה, יגרמו לכותבי XML רבים לאמר תודה ושלום ל-DTD, ולפנות את הדרך ל-XML Schema.

כדאי שתהיו שם כשזה יקרה.

נספח א'
מדריך מהיר ל-XML

לנוחיותכם הוספנו פרק זה, שמטרתו לרכז בקצרה, ולעשות סדר בנושאים העיקריים של XML, אשר סקרנו בספר.

החלקים המרכיבים את XML

כמתחיל ב-XML, תצטרך ללמוד שלושה נושאים חדשים:

1. XML בנוי היטב (Well formed XML)
2. DTD ו-XML חוקי
3. XSL

ניתן לומר ששלושת הנושאים הנ"ל מרכיבים יחד את הנושא הנקרא: XML.

XML בנוי היטב, הוא מסמך בסיסי המכיל את התגיות והנתונים ברמת התוכנית, ואשר נכתב על פי כללי המפרט של XML, כפי שמופיעים בהמלצת W3C. DTD הם ראשי תיבות של Document Type Definition, שהוא מסמך המרכז את תנאי המפרט של מסמך ה-XML. ב-DTD נפרט את שמות התגיות המותרות, תכונותיהן וסדר ההירכיה ביניהן, על פי דרישותינו.

XSL היא שפת גיליונות עיצוב, שרוב כוחה טמון, ביכולות הטרנספורמציה שלה. דרכה, מעבירים מסמך XML, על פי דרישות קלט שונות.

מתרגם (Parser) XML

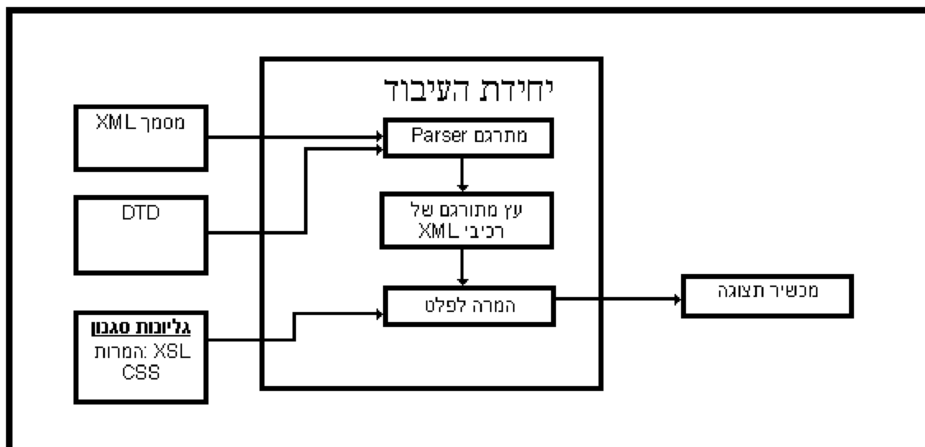
כדי שנוכל להשיג משהו מקובץ ה-XML שלנו, יש להעבירו דרך תוכנת תרגום (Parser), אשר תפקידה להפיק פלט מקובץ ה-XML על פי דרישתנו. מתרגמים של XML ניתן ליצור לבד, על פי הנחוץ לנו מהתוכנית, או להשתמש במתרגמים קיימים למשימות שונות. הדפדפן, לדוגמה, הוא מתרגם לכל דבר. בטעינת קובץ XSL אל הדפדפן, הוא יתרגם אותו ויציג אותו למשתמש. נסה לטעון מסמך XML ב-Internet Explorer 5 ותיווכח כי הדפדפן מציג אותו כ"עץ מתקפלי", זו ברירת המחדל שלו כמתרגם עבור קבצי XML המועברים שלא דרך גיליון סגנון. למעשה, מה שעושה הדפדפן במקרה זה, הוא הצגת נתוני המסמך בצורת עץ, על פי הסדר ההירכי של תגיותיו.

טרנספורמציית XSL

גיליון הסגנון XSL הינו מפרט מקובל להצגת סגנונות ב-XML, וכן ליצירת המרות עבור מסמכי XML. נכון לכתיבת שורות אלו, עדיין אין המלצה ברורה ל-XSL בעל סגנונות עיצוב, אלא רק עבור תכונות הטרנספורמציה (ההמרות) של XSL. מסיבה זו, הדפדפן היחידי שתומך ב-XML, Internet Explorer 5, תומך אך ורק בתכונת ההמרות של XSL. להשגת סגנונות ויזואליים (צבע גופן, צבע רקע, הזחת טקסט וכו׳) ניתן להשתמש בגיליונות סגנון מדורגים (CSS). ב-XSL נשתמש בעיקר עבור המרות.

כדי להבין את המושג המרות, נדמיין שברשותנו מסמך XML המפרט שמות ומאפיינים של כלי רכב. ה-DTD שלו, יפרט מה הן שמות התגיות בהן מותר להשתמש לייצוג תוכן במסמך ה-XML, אילו תכונות מותרות בכל תגית, ומה סדר ההיררכיה בין התגיות (למשל, אפשר להחליט ש-״חנות רכבים״ היא מרכיב השורש של המסמך, ומרכיב זה מכיל רכיבים בשם ״רכב״, המכילים רכיבים נוספים). תפקידה של XSL הוא להמיר את מסמך ה-XML לפלט נדרש על פי בקשתנו. למשל, כדי להציג את נתוני המסמך לרשת האינטרנט, יש ליצור מסמך XSL מתאים שידאג להמיר את נתוני ה-XML ל-HTML. או, כדי להציג את תוכן המסמך בטלפונים סלולריים, או מכשירים נישאים, יש ליצור מסמך XSL אחר שתפקידו ליצור המרה של הנתונים לפרוטוקול WAP וכו׳

התרשים הבא מתאר את הקשר שבין המושגים הנ״ל:



תרשים 1

XML בנוי היטב (Well Formed XML)

XML בנוי היטב הינו מסמך XML אשר עומד בתנאי התחביר הקשיחים של המלצת W3C.

ואילו התנאים:

1. אסור שתגיות מקוננות יחפפו.
2. שמות התגיות חייבים להיות תלויי רישיות (Case-Sensitive).
3. כל התגיות חייבות להיסגר.
4. ערכי תכונות (Attributes) חייבים להיות מסומנים בגרשיים.
5. המסמך חייב להכיל רכיב אחד בדיוק, שיהווה את רכיב השורש של המסמך.

כתיבת XML

בחלקו העליון של מסמך XML מצויה ההכרזה על סוג המסמך:

```
<? xml version="1.0"? >
```

הכרזת ה-XML אופציונלית, אם כי על פי מפרט XML בנוי היטב, יש להשתמש בה תמיד.

חלקו המרכזי של המסמך מורכב מתגיות בעלות שמות של רכיבים. רכיב השורש - Root Element (ידוע גם כרכיב המסמך Document Element), מכיל את כל תגיות הרכיבים:

```
<book>
    <section>
        <title>Harry Poter 1</title>
        <description>A nice book</description>
    </section>
    <section>
        <title>Harry Poter 2</title>
        <description>Another nice book</description>
    </section>
    <section>
        <title>Harry Poter 3</title>
        <description>One more nice book</description>
        <bigCover />
    </section>
</book>
```

כפי שידוע ב-HTML, תגיות נכתבות בתוך סימני קטן מ- (<) וגדול מ- (>). תגית הסגירה נכתבת כשלפני שם התגית, מצוי תו הלוכסן (/). הטקסט המוכל, הינו הטקסט שבין תגית הפתיחה ותגית הסגירה. (תגיות אלו נקראות "תגיות מכולה")

```
<title>Harry Poter 3</title>
```

כלל שלישי במפרט הנכון עבור XML לעיל, קובע כי כל התגיות חייבות להיסגר על ידי תגית סוגרת. עם זאת, ייתכנו מקרים בהם נרצה להשתמש בתגיות בודדות, אשר אינן מכילות טקסט פנימי. ב-HTML התגית
 היא דוגמה לתגיות מסוג זה. תגיות אלו חייבות להיות מיוצגות ב-XML בנוי היטב על ידי לוכסן אחרי שם התגית. לכן, התגית
 ב-XHTML צריכה להיראות כך:
 (שים לב לרווח אחרי הלוכסן - זהו מנהג מקובל, אך אינו חובה). כך, בדוגמה לעיל, התגית <bigCover /> הינה תגית בודדת.

הוספת הערות

הערות בקוד ה-XML (וגם במסמך ה-DTD בהמשך) ניתן לרשום כפי שנהוג לסמן הערות ב-HTML בין שני הסימנים <!-- ו- -->

כפי שאתה רואה, הצגתי את המסמך הנ"ל בעזרת הזחות של טקסט. זוהי צורת הצגה, שאינה חובה, אבל מומלצת ביותר, לשם הבנת סדר ההיררכיה של רכיבי המסמך.

הוספת סעיפי CDATA

ניתן להוסיף תגיות CDATA אשר יכולות להכיל טקסט, שאינו XML, ואינו ניתן לעיבוד על ידי מעבד ה-XML. הוספת הטקסט תיעשה בתוך תגית CDATA בין הכיתוב <![CDATA[לסגירה]]> כמודגם:

```
<!CDATA[
this abstract text area will not be parsed...It's just text...no meaning
]]>
```

DTD ו- XML חוקי

DTD - Document Type Definition הינו קובץ הגדרות אשר מתאר את המפרט החוקי של מסמך XML. בתוך הגדרת ה-DTD, יש לציין מהן שמות התגיות המותרות בקובץ ה-XML, מהו סוג הנתונים שכל תגית יכולה להכיל, מה סדר ההיררכיה שבין הרכיבים במסמך וכו'

XML חוקי - הינו מסמך XML אשר עומד במפרט ההגדרות של קובץ הגדרת מסמך DTD.

DTD **-** ניתן לכתוב בשתי דרכים:

1. באותו קובץ של מסמך ה-XML – (Internal DTD Subset). במקרה זה הצהרות ה-DTD ימוקמו בתחילת קובץ ה-XML.
2. כקובץ נפרד – (External DTD Subset). במקרה זה יש ליצור קובץ נפרד, אשר יכיל את הגדרות ה-DTD, ובראש מסמך ה-XML, יש ליצור הפניה לקובץ ההגדרות.

בניית DTD

כל מסמך DTD מורכב מ-״הגדרת סוג מסמך״ – אשר משמשת להגדרת שם המסמך. יתר ההגדרות, מוכלות בתוך הגדרה זו, כמודגם בתרשים:

```
<? XML Version="1.0" ?>
<!DOCTYPE books [
הגדרות לכל רכיב ורכיב
הגדרות לכל רכיב ורכיב
הגדרות לכל רכיב ורכיב
]>
```

כפי שאתה רואה, שם המסמך מצוי בתוך הגדרת ה-DOCTYPE, ולאחריו, סוגריים מרובעים, אשר מכילים את יתר ההגדרות הנדרשות עבור המסמך.

כפי שציינתי לעיל, ניתן לכתוב DTD, כחלק ממסמך ה-XML, או כקובץ חיצוני. הדרך המוצגת בסעיף זה, תואמת את הדרך הראשונה. כדי ליצור DTD חיצוני, יש ליצור הצהרת DOCTYPE בתוך מסמך ה-XML אשר מובילה למסמך הגדרות ה-DTD.

להלן דוגמה להצהרה כזו:

```
<!DOCTYPE books SYSTEM "MyDTD.dtd">
```

שורה זו מצהירה כי מסמך ה-XML בנוי על פי ההגדרות המצויות בקובץ נפרד בשם: "MyDTD.dtd"

במקרה זה, קובץ זה יכיל רק את יתר ההצהרות, ללא צורך בהצהרת DOCTYPE נוספת.

הוספת רכיבים להגדרת המסמך

לאחר שהצהרנו על שם המסמך, הגדרת רכיב במסמך נעשית על ידי הצהרת ELEMENT, המלווה בשם הרכיב, ובאפשרויות התכולה שלו:

```
<!ELEMENT book (section+)>
```

הצהרה זו, למשל, מייצרת רכיב מסמך חדש בשם book, אשר מכיל רכיב section, אחד או יותר.

בתוך הסוגריים, הצמודות לשם הרכיב, יצוינו שמות הרכיבים, או סוג המידע אשר רכיב זה (תגית זו) יכול להכיל.

הגדרת תוכן הרכיב

ברשותנו מספר דרכים להגדרת תוכנם של רכיבים:

1. הגדרת תוכן ״ריק״ – ניתן להגדיר תוכן ריק על ידי השימוש במילה EMPTY, באופן הבא:

```
<!ELEMENT br EMPTY>
```

2. בדוגמה הנ״ל יצרנו תגית בשם br, אשר אינה יכולה להכיל תוכן.

3. הגדרת תוכן כללי – ניתן להגדיר רכיב, אשר יכול להכיל כל סוג של תוכן חוקי. כלומר, רכיב זה יכול להכיל כל סוג של רכיבים בנים, כטקסט, ללא כל סוג של הגבלה:

```
<!ELEMENT div ANY>
```

בדוגמה זו יצרתי רכיב בשם div, אשר יכול "להכיל הכל".

4. הגדרת תוכן הירכי – ניתן להגדיר את הסידור ההירכי של רכיבים בנים לרכיב, בעזרת שימוש באופרטורים מתאימים. בדרך זו, נדון מייד.

שימוש באופרטורים

בהגדרת הרכיב שלעיל, מופיע סימן הפלוס (+) בצמוד לשם הרכיב המוכל. סימן זה בא לציין כי הרכיב section הינו הכרחי ויכול להופיע לפחות פעם אחת, או יותר. סימן הפלוס (+) קרוי אופרטור.

ישנם אופרטורים נוספים, בהם ניתן להשתמש בהגדרות ה-DTD, נלמד עליהם מתוך הדוגמאות הבאות:

```
<!ELEMENT section (title , subtitle? , (section | div)*)>
```

דוגמה זו מכילה מספר אופרטורים. ראשית, סימן הפסיק (,) המפריד בין הרכיב title לרכיב subtitle והביטוי (section | div)*. תפקידו של **סימן הפסיק (,)** הוא לתאר סדר בין רכיבים, על פי סדר הגדרתם. בדוגמה זו, רכיב title יבוא לפני רכיב subtitle אשר יבוא לפני הרכיב המורכב שלאחריו.

לאחר מכן, **סימני הסוגריים ()**, אשר מכילים בתוכם את הביטוי: section | div. סימני הסוגריים (), כמו תפקידם בחשבון, מציינים כי זהו ביטוי אחד. **סימן ה-Pipe (|)**, מציין תנאי לוגי מסוג "או". כלומר: הביטוי כולו אומר: רכיב מסוג section או רכיב מסוג div. לאחר מכן, מחוץ לסוגריים נמצאת **הכוכבית (*)**, אשר באה לציין כי הביטוי שבסוגריים כולו, הינו אופציונלי, ויכול להופיע פעם אחת או יותר.

סימן השאלה (?), הצמוד לשם הרכיב subtitle, בא לומר כי התגית subtitle הינה אופציונלית, ויכולה להופיע פעם אחת, לכל היותר.

לסיכום, משמעו של כל הביטוי הנ"ל הינו: "רכיב section מכיל רכיב מסוג title ולאחריו, אפס או רכיב אחד מסוג subtitle, ולאחריו יכול להופיע –רכיב מסוג section או רכיב מסוג div,והוא יופיע פעם אחת או יותר".

אופרטורים לייצוג תוכן של רכיבים

תו	תפקיד
+	רכיב הכרחי. יכול להופיע פעם אחת או יותר
*	רכיב אופציונלי. יכול להופיע פעם אחת או יותר
?	רכיב אופציונלי. יכול להופיע פעם אחת בלבד
|	רק רכיב אחד יכול להופיע
,	הרכיבים יופיעו על פי סדר הגדרתם

הגדרת תוכן מידע

כל ההגדרות הנ״ל טיפלו במבנה ההיררכי של מסמך ה-XML. כדי להגדיר רכיב (תגית) אשר מכיל טקסט, יש להשתמש במילה השמורה PCDATA.

למשל, הדוגמה הבאה מכריזה על רכיב div, אשר יכול להכיל תגית נוספת מסוג div, או טקסט:

```
<!ELEMENT div (div | #PCDATA)>
```

הגדרת תכונות (Attributes) לרכיבי המסמך

לאחר שיצרנו את רכיבי המסמך, נוכל להגדיר להם תכונות.

להגדרת תכונות לרכיבים, נשתמש בהגדרת ATTLIST, כשלאחריה, שם הרכיב, שם התכונה, סוג המידע אשר אותה תכונה יכולה להכיל בסוגריים, וערך ברירת המחדל של אותה תכונה. הגדרת ה-ATTLIST יכולה להכיל רשימה של כל התכונות האפשריות, בזו אחר זו, כמודגם להלן:

```
<!ATTLIST Car
   level  CDATA #REQUIRED
   name CDATA #IMPLIED
   color CDATA "blue"
>
```

בדוגמה הנ״ל הגדרנו שלוש תכונות לרכיב Car, וכן את סוג המידע של כל תכונה, והצהרת ברירת המחדל.

סוג המידע של תכונה

הגדרת סוג המידע של תכונה, מגדירה את סוג הערך אשר יכולה התגית לקבל במסמך ה-XML.

ישנן שלוש דרכים שונות להגדרת סוג המידע של תכונה:

1. ערך מחרוזתי – ניתן לקבוע ערך ברירת מחדל מחרוזתי, לפי כל מחרוזת הנתונה בין סימני גרשיים. (בדוגמה הקודמת, קבענו ערך "blue" לתכונה color)
2. סוג קבוע Tokenized – ניתן לקבוע ערך ברירת מחדל על פי סוגים קבועים מראש. את הסוגים אפרט מיד.
3. סוג מוגדר Enumerated – ניתן להגדיר סוגים במסמך ה-DTD, ולאחר מכן, להשתמש בהם. אדגים זאת בהמשך.

סוגים קבועים Tokenized

ישנם מספר סוגים קבועים אשר ניתן להשתמש בהם להצהרות ברירת מחדל עבור ערכי תכונות. הטבלה הבאה מרכזת את הסוגים הקיימים:

מילה שמורה	הסבר
ID	ערכה של התכונה עבור כל רכיב, חייב להיות ייחודי. לא יתכנו שני רכיבים בעלי אותו ערך. (ניתן להשוות זאת ל"מפתח-ראשי" המוכר מבסיסי נתונים טבלאיים)
IDREF	ערכה של התכונה מתייחס לערך ה-ID של רכיב אחר.
IDREFS	ערכה של התכונה מתייחס למספר רכיבים אחרים.
ENTITY	ערך התכונה חייב להיות שם של יישות קיימת. (על יישויות אסביר בעמודים הבאים)
ENTITIES	ערך התכונה יכול להיות מורכב משמות של יישויות שונות קיימות.
NMTOKEN	זוהי מילה המורכבת ממספר אותיות, ספרות, נקודות (.), מקפים (-), קו-תחתי (_), וכן, נקודותיים (:), כל עוד הן אינן בתחילת השם.
NMTOKENS	מספר מילים אשר מכילות את התווים הנ"ל.

הדוגמה הבאה, מצהירה על תכונה בשם SerialNum של הרכיב book, זוהי תכונה אשר ערכה חייב להיות בלעדי לכל רכיב book. כלומר, לא ייתכנו שני רכיבי book, אשר לשניהם אותו ערך SerialNum.

```
<!ATTLIST book SerialNum ID #REQUIRED>
```

סוגים מוגדרים Enumerated

ניתן לקבוע אילו ערכים יכולה תכונה לקבל.

בדוגמה הבאה, אנו מגבילים את ערכי התכונה color של הרכיב Car למחרוזות "red", "green" או "blue". (נאמר שרכיב Car מייצג קו מכוניות, אשר משווקות רק בצבעים אלו), בנוסף, אם התכונה color אינה נרשמת לצד התגית color, במסמך ה-XML, יהיה ערך ברירת המחדל שלה "red":

```
<!ATTLIST Car color (red | green | blue) "red">
```

הצהרת ערך ברירת המחדל של תכונה

הצהרת ברירת המחדל היא החלק השלישי בהצהרת ה-ATTLIST.

כפי שהבחנת, בהצהרות ברירת המחדל, השתמשתי במילים שמורות, או בערכים אחרים.הערכים האפשריים הם:

#FIXED – הערך הינו קבוע ואינו ניתן לשינוי. קביעת ערך השונה מהערך המוצהר, אינה חוקית.

```
<!ATTLIST Car
   speed CDATA #FIXED "fast"
>
```

#REQUIRED – התכונה נדרשת. המסמך אינו חוקי אם התכונה אינה מופיעה.

```
<!ATTLIST Car
   SerialNum ID #REQUIRED
>
```

#IMPLIED – התכונה היא אופציונלית. ללא ציון התכונה המסמך עדיין חוקי. למרות זאת, אי הכנסת התכונה אמורה להוביל להודעת שגיאה של יחידת העיבוד.

```
<!ATTLIST Car
   size CDATA #IMPLIED
>
```

הדוגמה הבאה משלבת מספר הצהרות על תכונות עבור התגית Car:

```
<!ATTLIST Car
   SerialNum ID #REQUIRED
   size CDATA #IMPLIED
   speed CDATA #FIXED "fast"
   color ( red | green | blue ) "red"
>
```

הצהרה על יישויות Entities

יישות היא שם המייצג מידע, אשר ניתן להשתמש בו בכל מקום במסמך, על ידי השימוש בשמו. היישויות הם למעשה קיצורי דרך, או תחליפי טקסט אשר באים לייצג טקסט גדול יותר. כך תוכל להגדיר שם יישות למידע או טקסט החוזר על עצמו מספר פעמים במסמך, ולהשתמש בו כקיצור בכל פעם שתזדקק לכך.

כפי שב-HTML נעשה שימוש ביישויות כמו   או © המייצגים רווח וסימן זכויות שמורות (©), גם ב-XML ניתן להשתמש ביישויות.

קיימים שני סוגים של יישויות:

1. Entity Reference - יישות לשימוש במסמך ה-XML – תחליף טקסט אשר ניתן להשתמש בו בתוך מסמך ה-XML עצמו. במקרה זה, יש להשתמש בשם היישות המוצמדת לסימן האמפרסנד (&) ובסופה סימן נקודה פסיק (;)

&EntityName;

2. Parameter Entity - יישות לשימוש במסמך ה-DTD – תחליף טקסט אשר ניתן להשתמש בו בתוך הגדרות ה-DTD. במקרה זה, יש להשתמש בשם היישות המוצמדת לסימן האחוזים (%) ובסופה סימן נקודה פסיק (;)

%EntityName;

את היישויות אנו יוצרים בתוך ה-DTD בעזרת ההצהרה ENTITY, מלווה בשם הקיצור הרצוי, והטקסט אותו הוא מחליף:

```
<!ENTITY Comp "The Blue and Red Cars Company for Israel Ltd.">
```

בדוגמה זו, הכרזנו על יישות לשימוש במסמך ה-XML. בכל פעם שבה ייעשה שימוש בקיצור &Comp; – תוחלף מילת הקיצור באותו מיקום במסמך במחרוזת הארוכה:

"The Blue and Red Cars Company for Israel Ltd."

כדי להכריז על יישות, לשימוש חוזר בתוך מסמך ה-DTD, יש להוסיף את סימן האחוז (%) לפני שם היישות בתוך הגדרת ה-Entity, באופן הבא:

```
<!ENTITY % text "#PCDATA | b | I | u | sub | sup | tab | spacerun | char">
```

דוגמה זו לקוחה מפרק 3 בספר, המדבר על DTD. הדוגמה תיצור קיצור עבור כותב ה-DTD. מכיוון שלאורך מסמך DTD קיים שימוש חוזר על הכתיבה:

#PCDATA | b | I | u | sub | sup | tab | spacerun | char

יצרנו יישות בתוך מסמך ה-DTD בשם text, וכעת, בכל פעם שנשתמש בקיצור %text; יישתל באותו המקום הטקסט החלופי הארוך יותר (**בתוך** ה-DTD).

לשם הבהרה, בהסתמך על הצהרת ה-ENTITY לעיל - מודגם המשפט הבא:

<!ELEMENT title (% **text;**)*>

המשפט זהה בדיוק למשפט הבא:

<!ELEMENT title (#**PCDATA | b | I | u | sub | sup | tab | spacerun | char**)*>

שימוש ביישויות ISO

כמו ב-HTML, גם ב-XML ניתן להשתמש ביישויות ISO קיימות. כך, ניתן לייצג תווים נסתרים, כמו סימן זכויות שמורות (©), סימן גדול מ- (>) אשר נעשה בו שימוש בייצוג תגיות.

נספח ג׳ - טבלאות נתונים, מציג את הקיצורים הקיימים של ISO.

שימוש בגיליונות סגנון

כאמור, כדי להציג את תוכן מסמך ה-XML כפלט, יש להעבירו דרך גיליון סגנון, אשר יקבע כיצד יוצגו תכני המסמך על גבי מכשיר התצוגה.

לצורך כך, קיימות שתי שפות סגנון:

1. CSS - **C**ascading **S**tyle **S**heets – שפת גיליונות מסוגננים
2. XSL - e**X**tensible **S**ty**L**e sheets – שפת גיליונות הניתנים להרחבה

עד היום, עדיין לא פורסמה המלצה לתקני סגנון של XSL לתצוגה. XSL משמש כיום לשינוי מסמך XML למסמך XML אחר, בעזרת תכונת הטרנספורמציה שלו.

לכן, כדי להציג מסמכי XML באופן ויזואלי על גבי הדפדפן, משתמשים כיום ב-CSS, אשר קובע מה תהיה תצורתו הוויזואלית של המסמך.

כדי להפוך מסמך XML למסמך XML שונה (למשל, למסמך XHTML שהוא מסמך HTML התואם לדרישות המפרט של XML בנוי היטב) משתמשים בטרנספורמציית XSL.

שימוש ב-CSS

לא אסביר כאן את השימוש ב-CSS, מכיוון שקצרה היריעה מלפרט זאת. תוכל לקרוא על כך לעומק בספר "HTML 4 למפתחי אתרים באינטרנט" בהוצאת "הוד עמי".

עם זאת, אדגים כיצד ניתן להשתמש בגיליון סגנון CSS כדי להציג נתוני מסמך XML על גבי הדפדפן, בהנחה שיש בידיך הידע בהגדרת סגנונות CSS.

ראשית, יש ליצור מסמך CSS, ולאחר מכן, לקשר בין קובץ ה-XML לקובץ הגדרות ה-CSS, על ידי ההצהרה הבאה, במסמך ה-XML:

```
<?xml-stylesheet type="text/css" href="FilePath"?>
```

הצהרה זו תבוא, לרוב, אחרי הצהרת ה-XML. כאשר במקום המילה FilePath יש להכניס את המיקום היחסי, ה-URL בו שמור קובץ הגדרות ה-CSS.

מבנה קובץ ה-CSS יהיה מורכב מהגדרות שונות, המורות לדפדפן כיצד יש להציג כל רכיב ורכיב בדפדפן.

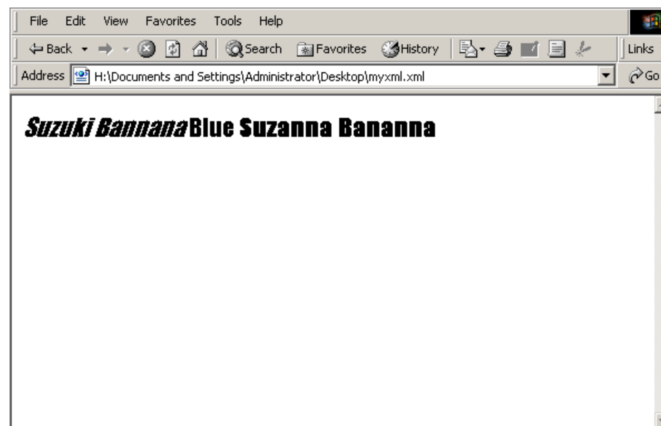
לדוגמה, הגדרות ה-CSS הבאות, מתאימות למסמך ה-XML הבא אחריהן:

```
<style type="text/css">
<!--
CAR
{
	font-size:22px;
	margin-left:20px;
	font-family:Impact
}
NAME
{
	font-style:italic;
}
OWNER
{
	font-weight:800;
}
-->
</style>
```

מסמך ה-XML הבא מקושר לקובץ ה-CSS הנ"ל (נניח ששמו Styles.css)

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="styles.css"?>
<CAR>
  <NAME>Suzuki Bannana</NAME>
  <COLOR>Blue</COLOR>
  <OWNER>
      <FIRSTNAME>Suzanna</FIRSTNAME>
      <LASTNAME>Bananna</LASTNAME>
  </OWNER>
</CAR>
```

התוצאה על גבי הדפדפן תיראה כך:



תרשים 2

כלומר, הדפדפן מציג את תוכנה של כל תגית במסמך XML על פי הגדרות הסגנון של מסמך ה-CSS.

שימוש ב-XSL

CSS מאפשרת לנו לקבוע עיצובים ויזואליים וסגנוניים עבור כל רכיב ורכיב במסמך ה-XML. עם זאת, CSS מוגבלת בכך, שאינה מאפשרת קביעת מבנה המסמך מחדש על פי דרישותינו. כך, לא ניתן להציג סדר על פי רצוננו, השמטת תגיות, הכפלת תגיות, מיון התגיות וכו׳

למזלנו, למטרות אלו קיימת XSL. והתכונה החשובה ביותר שלה היא ״טרנספורמציה״. תכונה זו מאפשרת להמיר מסמך XML למסמך XML חדש, הבנוי על פי רצוננו.

בדומה לעבודה עם CSS, כדי להעביר את מסמך ה-XML דרך גיליון סגנון XSL, יש ליצור ראשית את מסמך ה-XSL, ולאחר מכן, לקשר בין הקבצים, על ידי ההצהרה הבאה, במסמך ה-XML :

```
<?xml-stylesheet type="text/xsl" href="FilePath"?>
```

הצהרה זו תבוא, לרוב, אחרי הצהרת ה-XML. כאשר במקום המילה FilePath יש להכניס את המיקום היחסי, ה-URL בו שמור קובץ ה-XSL.

כתיבת XSL

XSL בנוי כך, שהוא מכיל תבניות. כל תבנית מכילה מידע להצגה על רכיב עץ אחר במסמך ה-XML.

כפי שציינתי קודם, אחד השימושים של XSL הוא המרת מסמך XML למסמך XHTML, שהוא מסמך HTML תקני, ויחד עם זאת הוא גם מסמך XML בנוי היטב. כך, למעשה, ניתן ליצור מסמך XHTML מתוך נתוני XML, אשר ניתן להציגן על גבי הדפדפן.

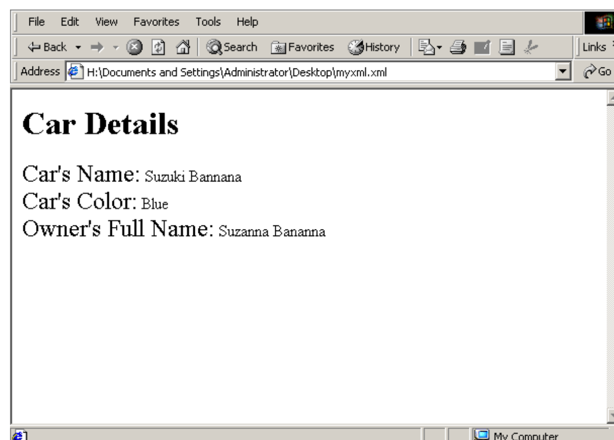
לדוגמה, המסמך הבא, הינו מסמך XML אשר מועבר דרך מסמך ה-XSL שבהמשך. ה-XSL מבצע טרנספורמציה על איברי ה-XML, ועל ידי ״תבניות התאמה״ (מיד אסביר) הוא יוצר מסמך XHTML הניתן לצפייה על גבי הדפדפן :

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="XSLDoc.xsl"?>
<CAR>
  <NAME>Suzuki Bannana</NAME>
  <COLOR>Blue</COLOR>
  <OWNER>
      <FIRSTNAME>Suzanna</FIRSTNAME>
      <LASTNAME>Bananna</LASTNAME>
  </OWNER>
</CAR>
```

הקוד הבא הינו מסמך XSL בשם קובץ "XSLDoc.xsl":

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
    <xsl:template match="/">
        <h1>Car Details</h1>
        <font size="+2">Car's Name:</font>
        <xsl:value-of  select="CAR/NAME" />
        <br />
        <font size="+2">Car's Color:</font>
        <xsl:value-of  select="CAR/COLOR" />
        <br />
        <font size="+2">Owner's Full Name:</font>
        <xsl:value-of  select="CAR/OWNER" />
        <br />
    </xsl:template>
</xsl:stylesheet>
```

תוצאות טעינת מסמך ה-XML בדפדפן Explorer 5 ייראו כפי המודגם בתרשים הבא:



תרשים 3

הסבר

כל מסמך XSL הינו גם מסמך XML בנוי היטב וחוקי. כך ש-XSL מורכבת מתגיות כמו כל מסמך XML. מכיוון ש-XSL הינה מסמך XML, יש להצהיר בראש המסמך על הצהרת ה-XML:

<?xml version="1.0"?>

התגית הראשונה אשר מציינת כי זהו מסמך XSL הינה תגית השורש של המסמך:

<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

תגית זו מציינת, כי זהו מסמך XSL והיא מפנה ל-namespace של XSL באתר W3C. (ראה פרק 4 בספר בו מוסבר נושא ה-namespace).

לאחר מכן, ניתן להשתמש ברכיבי XSL לשם יצירת טרנספורמציה של מסמך ה-XML למסמך XML חדש. בדוגמה שלעיל, יצרתי שינוי של מסמך ה-XML למסמך XHTML חדש זמני (בזיכרון), אשר ניתן להצגה על גבי הדפדפן.

התגית xsl:template

התגית הראשונה בה נעשה שימוש רב בגיליונות XSL, היא תגית התבנית (Template):

```
<xsl:template match="/">
```

תגית זו מכילה איזור אשר מכיל התאמות (Matches) לרכיבי מסמך ה-XML השונים. סימן ה-"/" מציין כי ההתאמה המבוצעת בתחום התגית הינה למסמך ה-XML כולו. במקרה זה, תתבצע ההתאמה רק לתגית הראשונה במסמך ה-XML. כדי להכיל את ההתאמה על כל התגיות יש להשתמש בתגית xsl:for-each או בתגית xsl:apply-templates בהן אדון בסעיפים הבאים.

בתוך תגית זו הכנסתי תגיות XHTML על פי הסדר בו רציתי שיעוצב המסמך. שים לב, שמבנה התגיות שונה מעט מהמבנה אליו התרגלת בכתיבת HTML. מהסיבה שזהו אינו מסמך HTML כי אם XHTML, והוא XML בנוי היטב (לכן, כל התגיות סגורות, ותגיות בודדות כמו br נראות כך:
).

התגית xsl:value-of

בין התגיות המוכרות, ודאי זיהית תגיות חדשות בשם xsl:value-of, תגיות אלו "שותלות" את התוכן של הרכיב המצוין בערך של תכונת ה-select שלהן, לדוגמה:

```
<xsl:value-of  select="CAR/COLOR" />
```

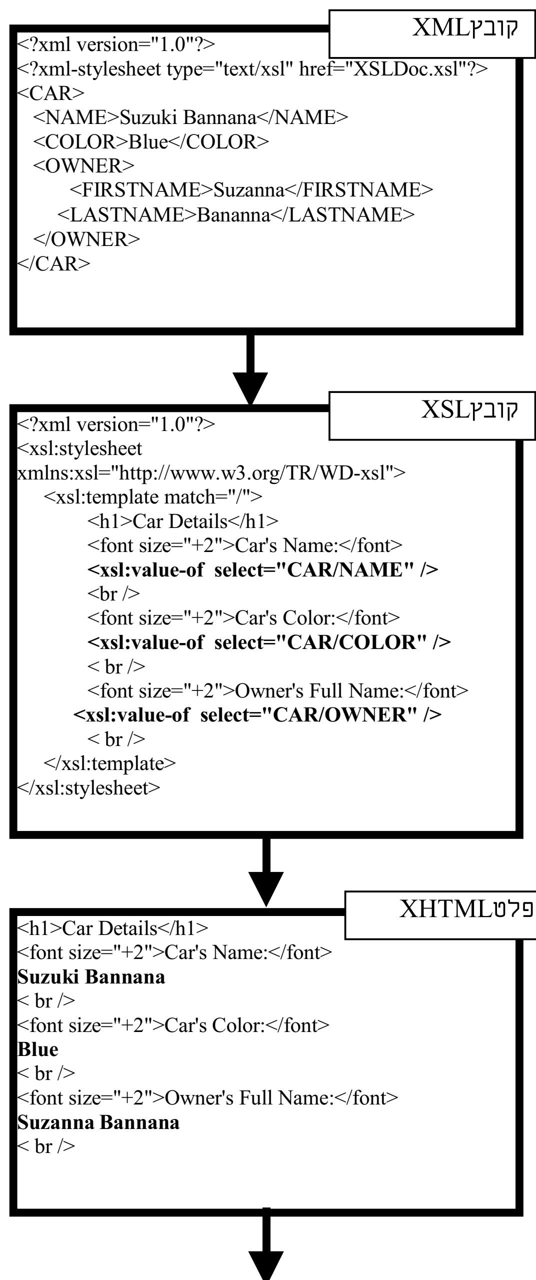
התגית הנ"ל מביאה להכנסת הערך של הרכיב color, אשר הינו רכיב בן של Car, באיזור בו מופיעה התגית.

שים לב לשימוש בסימני הקו הנטוי (/) אשר מייצגות שיוך היררכי של הורה-בן בין התגיות.

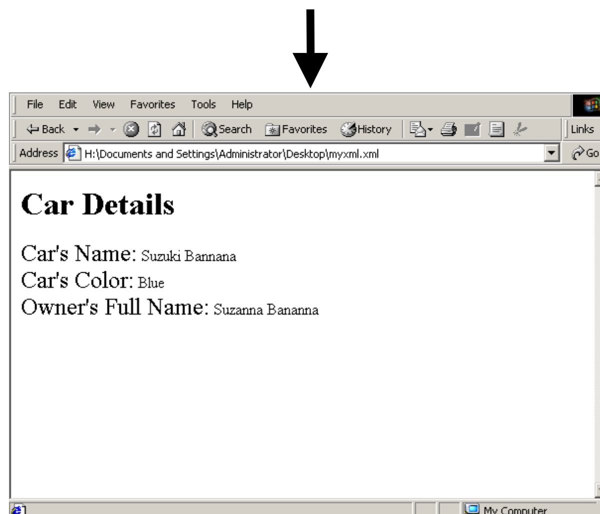
ניתן גם לגשת לערכים של תכונות של רכיבים. כדי לעשות כן, יש לצרף סימן קו נטוי (/) ולהצמיד לשם התכונה את סימן השטרודל (@). לדוגמה, התגית הבאה "שותלת" את ערכה של התכונה height של תגית בשם Image:

```
<xsl:value-of select="IMAGE/@height"/>
```

התרשים הבא מתאר את הטרנספורמציה המתבצעת כאשר מועבר מסמך ה-XML מן הדוגמה הקודמת, דרך מסמך ה-XSL ונפלט כ-XHTML. האיזורים המודגשים, מציגים את האיזורים במסמך ה-XSL לאחר שהוחלפו בטקסט שבמסמך ה-XML.


קובץXML
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="XSLDoc.xsl"?>
<CAR>
<NAME>Suzuki Bannana</NAME>
<COLOR>Blue</COLOR>
<OWNER>
<FIRSTNAME>Suzanna</FIRSTNAME>
<LASTNAME>Bananna</LASTNAME>
</OWNER>
</CAR>
קובץXSL
<?xml version="1.0"?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<h1>Car Details</h1>
Car's Name:
<xsl:value-of select="CAR/NAME" />

Car's Color:
<xsl:value-of select="CAR/COLOR" />
< br />
Owner's Full Name:
<xsl:value-of select="CAR/OWNER" />
< br />
</xsl:template>
</xsl:stylesheet>
פלטXHTML
<h1>Car Details</h1>
Car's Name:
Suzuki Bannana
< br />
Car's Color:
Blue
< br />
Owner's Full Name:
Suzanna Bannana
< br />



תרשים 4

שימוש בתגית xsl:for-each להכלת תבנית על מספר רכיבים

בשיטה בה השתמשנו עד כה, הציג הדפדפן רק את נתוני הרכיב הראשון במסמך ה-XML. כך, למשל, אם היינו מעבירים את הקובץ הבא דרך גיליון הסגנון שבעמודים קודמים, היינו מקבלים את נתוני רכיב ה-Car הראשון בלבד:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="xsldoc.xsl"?>
<CARSHOP>
 <CAR>
  <NAME>Suzuki Bannana</NAME>
  <COLOR>Blue</COLOR>
  <OWNER>
     <FIRSTNAME>Suzanna</FIRSTNAME>
     <LASTNAME>Bananna</LASTNAME>
  </OWNER>
 </CAR>
 <CAR>
  <NAME>Apolo Baloni</NAME>
  <COLOR>Green</COLOR>
  <OWNER>
     <FIRSTNAME>David</FIRSTNAME>
     <LASTNAME>Barbunia</LASTNAME>
  </OWNER>
 </CAR>
```

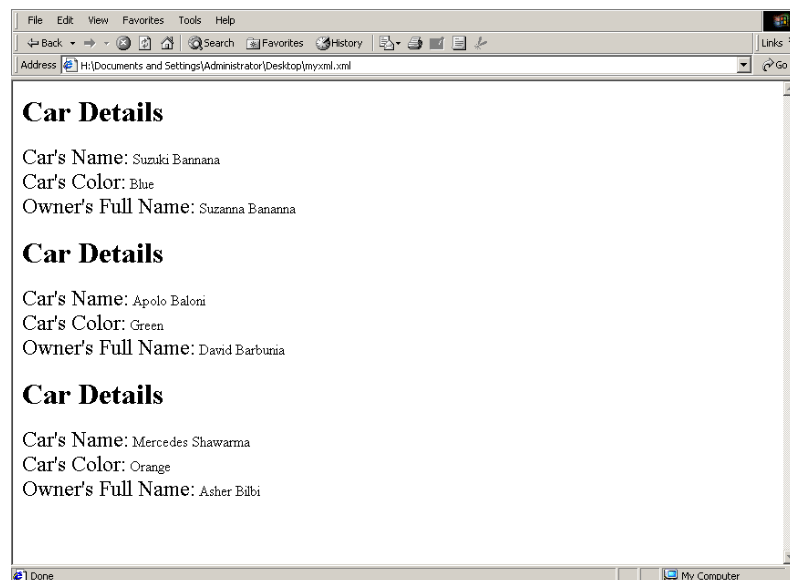
```
  <CAR>
    <NAME>Mercedes Shawarma</NAME>
    <COLOR>Orange</COLOR>
    <OWNER>
        <FIRSTNAME>Asher</FIRSTNAME>
        <LASTNAME>Bilbi</LASTNAME>
    </OWNER>
  </CAR>
</CARSHOP>
```

לכן, כדי להציג את כל הרכיבים בזה אחר זה, יש להשתמש בתגית xsl:for-each של XSL.

גיליון ה-XSL הבא, הינו אותו גיליון שהצגתי קודם, אלא שהפעם הוא יביא להצגת כל תגיות ה-Car על הדפדפן – שים לב לתגית המודגשת:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <xsl:template match="/">
   <xsl:for-each select="CARSHOP/CAR">
       <h1>Car Details</h1>
       <font size="+2">Car's Name:</font>
       <xsl:value-of select="CAR/NAME" />
       <br />
       <font size="+2">Car's Color:</font>
       <xsl:value-of select="CAR/COLOR" />
       <br />
       <font size="+2">Owner's Full Name:</font>
       <xsl:value-of select="CAR/OWNER" />
       <br />
   </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

התרשים הבא מציג את תוצאות העברת קובץ ה-XML הנ"ל דרך מסמך ה-XSL:



תרשים 5

שימוש בתגית xsl:apply-templates

בדוגמאות הקודמות, ציינתי את הרכיב עליו נעשית ההתאמה בעזרת סימן הקו הנטוי (/). סימן זה מייצג את רכיב המסמך, ובכל גיליון XSL יש חובה להתייחס אליו.

עם זאת, שיטה זו מחייבת אותנו ביצירת התאמות אך ורק לרכיב השורש. ומה אם אנחנו לא מעוניינים בהתאמות לרכיב השורש? ומה אם נרצה התאמות לרכיבים מסוימים בלבד?

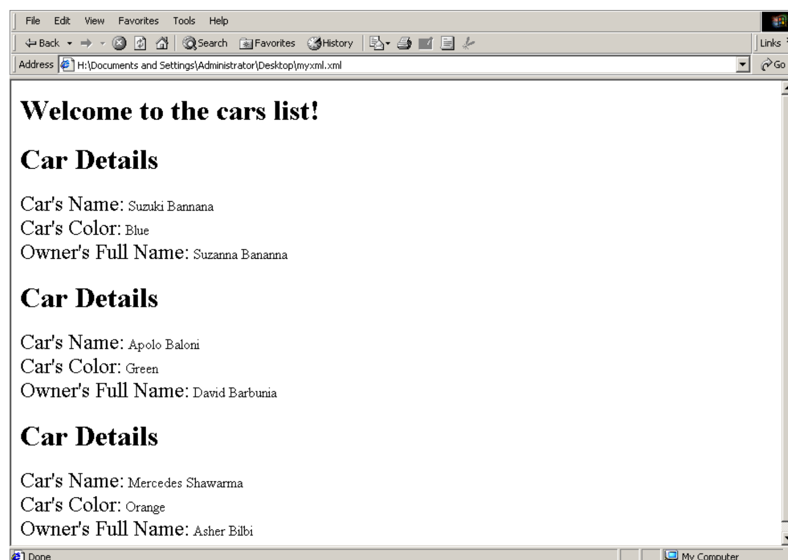
לשם כך קיימת התגית xsl:apply-templates אשר מאפשרת יצירת תבניות לרכיבים נבחרים. תגית זו הינה תגית בודדת ולכן, אין צורך לסגור אותה בתגית סוגרת. תפקידה הוא להורות לדפדפן להכיל תבנית template, אשר מוצגת במסמך XSL עבור כל רכיב התואם לערך התכונה select שלה.

הדוגמה הבאה, בנויה משתי תבניות template. האחת, עבור רכיב השורש, והשנייה עבור רכיבי ה-Car אשר הינם רכיבים בנים של הרכיב Carshop:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <xsl:template match="/">
  <h1>Welcome to the cars list!</h1>
     <xsl:apply-templates select="CARSHOP/CAR" />
  </xsl:template>
```

```
<xsl:template match="CARSHOP/CAR">
        <h1>Car Details</h1>
        <font size="+2">Car's Name:</font>
        <xsl:value-of select="NAME" />
        <br />
        <font size="+2">Car's Color:</font>
        <xsl:value-of select="COLOR" />
        <br />
        <font size="+2">Owner's Full Name:</font>
        <xsl:value-of select="OWNER" />
        <br />
  </xsl:template>
</xsl:stylesheet>
```

התבנית המוצגת עבור רכיב השורש, מכילה תגית XHTML של כותרת מסוג H1, וכן, תגית xsl:apply-templates אשר מורה לדפדפן "לשתול" בקטע זה (מיד אחרי תגית ה-H1) את התבניות עבור כל רכיב ה-Car. התבנית עבור כל רכיב Car ממוקמת בהמשך בתוך התגית xsl:template. התוצאה על גבי הדפדפן תיראה כך:



תרשים 6

סינון רכיבים על ידי שימוש בסוגריים מרובעים

ניתן לקבוע שתבנית מסוימת תחול רק על רכיב מסוים בקובץ ה-XML. כדי לעשות זאת, נשתמש בסוגריים מרובעים, בצמוד לשם רכיב האב.

אפשרות ראשונה - הגבלת תבנית לרכיב אשר קיים עבורו רכיב בן:

```
<xsl:template match="CAR[ROOF]">
```

בדוגמה זו, תתבצע התבנית, רק עבור רכיבי Car אשר קיים להם רכיב בן מסוג Roof.

אפשרות שנייה, שימוש באופרטורים השוואתיים :

```
<xsl:template match="CAR[ROOF='big']">
<xsl:template match="CAR[ROOF!='big']">
```

בדוגמה הראשונה תתבצע התבנית רק עבור רכיבי Car אשר מכילים רכיב בן מסוג Roof אשר מכיל את הטקסט "big" (נעשה כאן השימוש באופרטור ההשוואה "=").

בדוגמה השנייה, תתבצע התבנית עבור רכיבי Car אשר מכילים רכיב בן מסוג Roof אשר אינו מכיל את הטקסט "big" (נעשה כאן שימוש באופרטור "אי-שוויון" המיוצג על ידי סימן קריאה-שווה - "=!").

מיון רכיבים בעזרת order-by

תוכל לקבוע את סדר הרכיבים, על פיהם יוצג המידע על הדפדפן בעזרת תכונת order-by של תגיות xsl:apply-templates ו-xsl:for-each.

לדוגמה, התגית הבאה, ממיינת את רכיבי ה-Car של רכיב ה-Carshop בסדר עולה על פי ערכו של הרכיב Size :

```
<xsl:apply-templates select="CARSHOP/CAR" order-by="+SIZE">
```

סימן הפלוס (+) מייצג מיון עולה, והסימן מינוס (-) מיון יורד.

ניתן גם לקבוע סדר קדימויות למיונים. לשם כך, יש להפריד בין שמות השדות למיון על ידי נקודה-פסיק (;).

הדוגמה הבאה, קובעת שרכיבי STUDENT של CLASS ימוינו ראשית על פי ערכו של ID (נאמר, מספר סידורי של תלמיד) , אחר-כך, על פי ערכו של LASTNAME ואחר-כך על פי ערכו של FIRSTNAME :

```
<xsl:for-each select="CLASS/STUDENT" order-by="+ID; +LASTNAME; +FIRSTNAME">
```

בדיקת תנאים בעזרת התגית xsl:if

ניתן לקבוע בתוך תבנית, כי מידע מסוים יוצג אך ורק אם תנאי מסוים מתקיים.

כדי ליצור את התנאי, נשתמש בתגית <xsl:if> בעלת תכונה בשם test, האחראית על בדיקת התנאי.

כדי לבדוק תנאי שוויון מסוים, יש להשתמש בסוגריים מרובעים ([ ]) ובסימן נקודה ואחריה סימן שווה (=.) כמודגם :

```
<xsl:if test="COUNTRY[.='Israel']">He is from Israel!</xsl:if>
```

בדוגמה זו, אם מכילה התגית COUNTRY את הטקסט 'Israel' – יישתל הטקסט : !He is from Israel במקום הנדרש.

דוגמה נוספת :

```
<xsl:if test="CAR/@color[.!='red']">The car is not red!</xsl:if>
```

דוגמה זו, תגרום לשתילת הטקסט !The car is not red עבור רכיב CAR אשר ערך התכונה color שלו, שונה מ-'red'.

יצירת טבלת XHTML מנתוני מסמך XML

לסיכום פרק זה, אראה דוגמה לשימוש ב-XML וגיליון XSL להצגת טבלת נתונים בעזרת הדפדפן.

גיליון ה-XSL הבא, יוצר טרנספורמציה של מסמך ה-XML שאחריו, למסמך XHTML.

שים לב לבדיקת התנאי, אשר מציג בטבלה **רק** את הלקוחות ממין נקבה.

המיון נעשה על פי גילאי הלקוחות.

```
<?xml version="1.0"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

  <xsl:template match="/">
  <html>
   <head>
     <title>The Female Customers Page</title>
   </head>
   <body>
      <table border="3">
      <tr><th>Name</th><th>Age</th></tr>
      <xsl:apply-templates select="CUSTOMERS/CUSTOMER" order-by="AGE" />
      </table>
   </body>
  </html>
  </xsl:template>

  <!-- A template matching CUSTOMER Elements -->
  <xsl:template match="CUSTOMERS/CUSTOMER">
        <xsl:if test="GENDER[.='Female']">
             <tr>
                 <td><xsl:value-of select="NAME" /></td>
                 <td><xsl:value-of select="AGE" /></td>
             </tr>
        </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

הנה קובץ ה-XML אשר יועבר דרך גיליון ה-XSL הנ״ל:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="xsldoc2.xsl"?>
<CUSTOMERS>
  <CUSTOMER>
    <NAME>Amit Kasher</NAME>
    <GENDER>Male</GENDER>
    <AGE>22</AGE>
  </CUSTOMER>
```

```
  <CUSTOMER>
    <NAME>Suzanna Bannana</NAME>
    <GENDER>Female</GENDER>
    <AGE>31</AGE>
  </CUSTOMER>
  <CUSTOMER>
    <NAME>Hu Lo Po</NAME>
    <GENDER>Male</GENDER>
    <AGE>12</AGE>
  </CUSTOMER>
  <CUSTOMER>
    <NAME>Batya Ana</NAME>
    <GENDER>Female</GENDER>
    <AGE>67</AGE>
  </CUSTOMER>
  <CUSTOMER>
    <NAME>Gever Amiti</NAME>
    <GENDER>Male</GENDER>
    <AGE>120</AGE>
  </CUSTOMER>
  <CUSTOMER>
    <NAME>Ishi Moto</NAME>
    <GENDER>Male</GENDER>
    <AGE>66</AGE>
  </CUSTOMER>
  <CUSTOMER>
    <NAME>Bira Belira</NAME>
    <GENDER>Female</GENDER>
    <AGE>42</AGE>
  </CUSTOMER>
</CUSTOMERS>
```

להלן התוצאה על גבי הדפדפן:

Name	Age
Suzanna Bannana	31
Bira Belira	42
Batya Ana	67

תרשים 7

למעשה, יצרתי טבלת XHTML של כל הלקוחות ממין נקבה, הממוינות על פי גילן.

דוגמה זו ממחישה את השימוש של XML בשילוב עם גיליון XSL לייצוג תוכנו של מסמך על גבי הדפדפן, בצד הלקוח.



בדוגמה זו בחרתי להשתמש במעבר על כל רכיבי מסמך ה-XML, בעזרת תגית ה- xsl:apply-templates . יחד עם זאת, ניתן ליצור טבלה דומה על ידי שימוש בתגית xsl:for-each. אתגר זה השארתי לכם.

סקריפטים של XML DOM

מעבר לכתיבת מסמכי XML ו-XSL, ניתן לגשת אל רכיבי מסמך ה-XML בזמן ריצה, בצד הלקוח (דרך הדפדפן) או בצד השרת (עמודי ASP למשל). הגישה נעשית באמצעות מודל האובייקטים של המסמך, הוא ה-DOM (ראשי תיבות של Document Object Model).

מתוך ההנחה שאתה מכיר כבר את שפת JavaScript, בוודאי נתקלת במונח HTML DOM. ה-HTML DOM הינו מבנה היררכי של רכיבי מסמך HTML. ההבדל בין HTML DOM לבין XML DOM הוא בכך, ש-DOM של HTML קבוע מראש ומכיל אובייקטים קבועים בעלי תכונות ושיטות קבועות. מאחר ו-XML הינה שפה "ניתנת להרחבה" ואת רכיבי ה-XML אתה קובע, ה-DOM של מסמך ה-XML נקבע על פי מסמך ה-XML ורכיביו לעולם אינם ידועים מראש.

עם זאת, לכל מסמך XML קיים רכיב שורש, ולו רכיבים בנים, ולהם ייתכנו רכיבים בנים נוספים, תכונות וכו' – לכל אלו ניתן לגשת בעזרת סקריפטים אשר מופעלים על ה-DOM של המסמך. קיימים כמה אובייקטים מיוחדים לשם עבודה עם ה-DOM של XML ובראשם ה-DOMDocument המייצג מופע של מסמך DOM.

לא אכנס בנספח זה אל תוך פרטי ה-DOM, החל מפרק 3 בספר אנו דנים בכך לעומק ויוצרים שינויים למסמכים בעזרת DOM.

הגדרת מסמכים בעזרת XML Schema

מעבר ליכולת הגדרת המסמכים של DTD, קיימת שפה נוספת, חדשה יחסית, להגדרת מסמכי XML. שפה זו היא למעשה מסמך XML לכל דבר בעל תגיות יחודיות והיא נקראת XML Schema. מפרט ה-XML Schema מפותח על ידי Microsoft והוא מביא גישה חדשה לעבודה עם מסמכי XML חוקיים. הגישה של XML Schema שונה מאוד מ-DTD ואופן הכתיבה קל לקריאה ובעל ניידות גבוהה בהרבה מזו של DTD.

נושא ה-XML Schema נדון בהרחבה בפרק נפרד.

XML בצד הלקוח לעומת XML בצד השרת

אפשר לומר כי המשתמשים ב-XML המיועד לרשת האינטרנט נחלקים לשתי אוכלוסיות:

1. בוני אתרים ומעצבים אשר מעוניינים להרחיב את אפשרויות התצוגה שלהם על גבי הדפדפן – בצד הלקוח.
2. מפתחי יישומים בצד השרת, אשר מעוניינים להשתמש בצורת אחסון נתונים קלה וגמישה.

לדאבונם של חברי הקבוצה הראשונה, קיים רק דפדפן אחד כיום, אשר קורא XML וגיליונות XSL, והוא, כאמור, Internet Explorer. כך, שלעת עתה השימוש ב-XML בצד הלקוח מוגבל אך ורק לדפדפן זה (אבל טוב להתכונן לעתיד).

לקבוצה השנייה, ישנם כלים רבים בצד השרת לשימוש ב-XML, ביניהם אובייקטים של DOM, ובפרקים המרכזיים של הספר תקרא על כך בהרחבה.

סוף תקציר הוא רק התחלה

כאן מסתיים המדריך המהיר שלנו ל-XML. כאמור, במהלך הספר תלמד טכניקות נוספות וחשובות לעבודה עם XML שלא הוצגו כאן, וביניהן שימוש ב-XML DOM בצד הלקוח ובצד השרת, דרך עמודי ASP ו-VB וכן פרק ייחודי לנושא XML Schema.

השתדלנו להביא כאן את הנושאים המרכזיים ב-XML ברמה הבסיסית.

אם התחלת לקרוא נספח זה לפני שאר הפרקים בספר - אני מקווה שעזרתי לך להבין את הבסיס של XML ומאחל לך הנאה בהמשך הקריאה. אם קראת נספח זה לאחר שקראת את כל הספר, אני מקווה שעזרתי לך לארגן את הכל בראש מחדש ולשנן את הנושאים הללו לעומק. ושוב, בהצלחה עם XML!

נספח ב'
אתרים ברשת
העוסקים בנושאי XML

אתרים רשמיים

W3C

האתר הרשמי של הארגון העולמי W3C אשר אחראי לפרסום ההמלצות של XML, XSL, XHTML ונושאים נוספים:

http://www.w3c.org

Microsoft

אתר הבית של "מיקרוסופט" אשר מובילה את השימוש ב-XML ו-XML Schema:

http://www.microsoft.com

MSDN

אתר הרשת למפתחים של "מיקרוסופט". באתר זה תמצא רשימות עזרה וטבלאות עזר בכל הנושאים, גם ב-XML:

http://msdn.microsoft.com

http://msdn.microsoft.com/xml

אתרי עזרה

Web Developer

אתר המרכז מדריכים ויזואליים (Tutorials), ופורומים בנושאים רבים הקשורים בבניית אתרים. בין היתר, מכיל האתר חומר רב בנושא XML:

http://www.webdeveloper.com

XML.com

זהו אחד האתרים הגדולים ביותר הקיימים לנושאי XML. האתר מכיל מדריכים ויזואליים, מאמרים, טיפים, רשימות FAQ, פורומים, חדשות ועוד... זהו אתר מקצועי ומומלץ ביותר:

http://www.xml.com/default.asp

XMLU

אתר נוסף המרכז מאמרים והסברים רבים לגבי XML:

http://www.xmlu.com

Web monkey

אתר המרכז מדריכים ויזואליים, מאמרים, טיפים, ועמודי עזרה רבים בבניית אתרים בין היתר, תמצא הסברים רבים על HTML, CSS, DHTML, JavaScript ו-XML:

http://www.webmonkey.com

CNET Builder

אתר גדול המרכז מידע רב בכל תחומי הבניה לרשת. בין היתר תמצאו כאן עמוד נרחב בנושא XML:

http://www.builer.com

Project Cool Developer Zone

את המרכז מידע על מספר טכנולוגיות לבניית אתרים. הקישור כאן יוביל אל עמוד ה-XML של האתר, אשר מכיל מדריכים רבים ויעילים בנושא:

http://www.projectcool.com/developer/xmlz/

Web Reference

אתר נרחב לבוני אתרים. הקישור כאן מוביל לעמוד ה-XML, אשר מכיל מידע רב על XML וכמו כן, פורומים, קישורים ועוד:

http://www.webreference.com/xml

Web Review XML

אתר זה מכיל גם הוא מידע מרוכז ומקצועי על XML, כמו גם פורומים, מדריכים למתחילים ומתקדמים לגליונות סגנון ועוד:

http://webreview.com/wr/pub/XML

irt.org XML FAQ

אתר המרכז שאלות ותשובות (FAQ) בנושא XML, DTD, XSL וכו':

http://developer.irt.org/script/xml.htm

נספח ג'
טבלאות נתונים

בכל ספר רציני בנושא מחשבים, תמצא נספח המרכז את הנתונים בטבלאות זמינות ונוחות לקריאה.

אף אחד מאיתנו לא יכול בלי טבלאות, ואף אחד מאיתנו אינו זוכר את כל החומר בעל פה. בדרך כלל, כשאני זקוק לנתון מסוים, המקום הראשון שבו תמצאו אותי מחפש את מבוקשי, הוא באינדקס או בטבלאות שבסוף הספר.

חוקים עבור XML בנוי היטב (Well Formed XML)

- אסור שתגיות מקוננות יחפפו.
- שמות התגיות חייבים להיות תלויי רישיות (Case-Sensitive).
- כל התגיות חייבות להיסגר.
- ערכי תכונות (Attributes) חייבים להיות מסומנים בגרשיים.
- המסמך חייב להכיל רכיב אחד בדיוק, שיהווה את רכיב השורש של המסמך.

Document Type Definitions – DTD

פקודות DTD

תגית DTD	תפקיד	כתיבה לדוגמה
<!DOCTYPE>	הגדרת שם מסמך	<? XML Version="1.0" ?> <!DOCTYPE books [*הגדרות לכל רכיב ורכיב* *הגדרות לכל רכיב ורכיב* *הגדרות לכל רכיב ורכיב*]>
<!ELEMENT>	הגדרת רכיב	<!ELEMENT book (section+)>
<!ATTLIST>	הגדרת תכונות לרכיב	<!ATTLIST car level CDATA #REQUIRED name CDATA #IMPLIED color CDATA "blue" >
<!ENTITY>	הגדרת יישות חיצונית	<!ENTITY Comp "The Blue and Red Cars Company for Israel Ltd.">
	הגדרת יישות פנימית	<!ENTITY % text "#PCDATA | b | I | u | sub | sup | tab |spacerun |char">

אופרטורים לייצוג תוכן של רכיבים

תו	תפקיד
+	רכיב הכרחי. יכול להופיע פעם אחת או יותר
*	רכיב אופציונלי. יכול להופיע פעם אחת או יותר
?	רכיב אופציונלי. יכול להופיע פעם אחת בלבד
|	רק רכיב אחד יכול להופיע
,	הרכיבים יופיעו על פי סדר הגדרתם

סוגים קבועים Tokenized עבור תכונות

מילה שמורה	הסבר
ID	ערכה של התכונה עבור כל רכיב, חייב להיות ייחודי. לא ייתכנו שני רכיבים בעלי אותו ערך (ניתן להשוות זאת ל"מפתח-ראשי" המוכר מבסיסי נתונים טבלאיים).
IDREF	ערכה של התכונה מתייחס לערך ה-ID של רכיב אחר.
IDREFS	ערכה של התכונה מתייחס למספר רכיבים אחרים.
ENTITY	ערך התכונה חייב להיות שם של יישות קיימת.
ENTITIES	ערך התכונה יכול להיות מורכב משמות של יישויות שונות קיימות.
NMTOKEN	זוהי מילה המורכבת ממספר אותיות, ספרות, נקודות (.), מקפים (-), קו-תחתי (_), וכן, נקודותיים (:), כל עוד הן אינן בתחילת השם.
NMTOKENS	מספר מילים אשר מכילות את התווים הנ"ל.

גליון סגנון XSL

תגיות XSL Tranformations

תגית XSL	תפקיד	כתיבה לדוגמה
xsl:stylesheet	הגדרת גיליון XSL	<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
xsl:template	הגדרת תבנית התאמה	<xsl:template match="/"> Car Details Car's Name: <xsl:value-of select="CAR/NAME" /> </xsl:template>
xsl:apply-templates	הכלת תבניות	<xsl:apply-templates select="CARSHOP/CAR" />
xsl:value-of	שתילת ערך של רכיב	<xsl:value-of select="CAR/COLOR" />
	@-ערך תכונה	<xsl:value-of select="IMAGE/@Height" />
xsl:for-each	מעבר על כל הרכיבים	<xsl:for-each select="CARSHOP/CAR"> Car Details Car's Name: <xsl:value-of select="CAR/NAME" /> </xsl:for-each>
xsl:if	בדיקת תנאי	<xsl:if test="COUNTRY[.='ISRAEL']" >He is from Israel!</xsl:if>

Microsoft XML Schema

רכיבי XML Schema

שם רכיב	תיאור
<Schema>	רכיב השורש של מסמך ה-XML Schema
<ElementType>	מגדיר סוג של רכיב במסמך ה-XML
<AttributeType>	מגדיר סוג של תכונה במסמך ה-XML
<element>	מופיע בתוך תחום ההכלה של התגית <ElementType> כדי להגדיר רכיב בן לרכיב המוגדר בה.
<group>	מופיע בתוך תחום ההכלה של התגית <ElementType> כדי להגדיר כיצד רכיבים בנים מקובצים בה.
<attribute>	מופיע בתוך תחום ההכלה של התגית <AttributeType> כדי להגדיר תכונה לרכיב המוגדר בה.
<datatype>	מופיע בתוך תחום ההכלה של התגית <AttributeType> או התגית <ElementType> כדי להגדיר את סוג המידע המוכל בהן.
<description>	תגית אשר מספקת מידע עבור הרכיבים: <Schema>, <ElementType> ו- <AttributeType>

סוגי ערכים של XML Schema

bin.base64	MIME-style Base64 encoded binary BLOB.
bin.hex	Hexadecimal digits representing octets.
boolean	0 or 1, where 0 == "false" and 1 =="true".
char	String, one character long.
date	Date in a subset ISO 8601 format, without the time data. For example: "1994-11-05".
dateTime	Date in a subset of ISO 8601 format, with optional time and no optional zone. Fractional seconds can be as precise as nanoseconds. For example, "1988-04-07T18:39:09".
dateTime.tz	Date in a subset ISO 8601 format, with optional time and optional zone. Fractional seconds can be as precise as nanoseconds. For example: "1988-04-07T18:39:09-08:00".
fixed.14.4	Same as "number" but no more than 14 digits to the left of the decimal point, and no more than 4 to the right.
float	Real number, with no limit on digits; can potentially have a leading sign, fractional digits, and optionally an exponent. Punctuation as in U.S. English. Values range from 1.7976931348623157E+308 to 2.2250738585072014E-308.
int	Number, with optional sign, no fractions, and no exponent.
number	Number, with no limit on digits; can potentially have a leading sign, fractional digits, and optionally an exponent. Punctuation as in U.S. English. (Values have same range as most significant number, R8, 1.7976931348623157E+308 to 2.2250738585072014E-308.)
time	Time in a subset ISO 8601 format, with no date and no time zone. For example: "08:15:27".
time.tz	Time in a subset ISO 8601 format, with no date but optional time zone. For example: "08:1527-05:00".
i1	Integer represented in one byte. A number, with optional sign, no fractions, no exponent. For example: "1, 127, -128".
i2	Integer represented in one word. A number, with optional sign, no fractions, no exponent. For example: "1, 703, -32768".
i4	Integer represented in four bytes. A number, with optional sign, no fractions, no exponent. For example: "1, 703, -32768, 148343, -1000000000".
r4	Real number, with seven digit precision; can potentially have a leading sign, fractional digits, and optionally an exponent. Punctuation as in U.S. English. Values range from 3.40282347E+38F to 1.17549435E-38F.
r8	Real number, with 15 digit precision; can potentially have a leading sign, fractional digits, and optionally an exponent. Punctuation as in U.S. English. Values range from 1.7976931348623157E+308 to 2.2250738585072014E-308.
ui1	Unsigned integer. A number, unsigned, no fractions, no exponent. For example: "1, 255".
ui2	Unsigned integer, two bytes. A number, unsigned, no fractions, no exponent. For example: "1, 255, 65535".
ui4	Unsigned integer, four bytes. A number, unsigned, no fractions, no exponent. For example: "1, 703, 3000000000".
uri	Universal Resource Identifier (URI). For example, "urn:schemas-microsoft-com:Office9".
uuid	Hexadecimal digits representing octets, optional embedded hyphens that are ignored. For example: "333C7BC4-460F-11D0-BC04-0080C7055A83".

סוגי ערכים "פרימיטיביים"

מעבר לסוגי הערכים שהוגדרו על ידי מיקרוסופט, XML Schema תומכת גם בסוגים הבאים, אשר נוספו בהמלצת W3C עבור XML Schema:

entity	Represents the XML ENTITY type.
entities	Represents the XML ENTITIES type.
enumeration	Represents an enumerated type (supported on attributes only).
id	Represents the XML ID type.
idref	Represents the XML IDREF type.
idrefs	Represents the XML IDREFS type.
nmtoken	Represents the XML NMTOKEN type.
nmtokens	Represents the XML NMTOKENS type.
notation	Represents a NOTATION type.
string	Represents a string type.

ISO ישויות / ISO Entities

הסבר	תו	קוד יישות	מילה שמורה
horizontal tab			
line feed	
space			
exclamation mark	!	!	
double quotation mark	"	"	"
number sign	#	#	
dollar sign	$	$	
percent sign	%	%	
ampersand	&	&	&
apostrophe	'	'	
left parenthesis	(	(	
right parenthesis	)	)	
asterisk	*	*	
plus sign	+	+	
comma	,	,	
hyphen	-	-	
period	.	.	
slash	/	/	
digits 0-9		0-9	
colon	:	:	
semicolon	;	;	
less-than sign	<	<	
equals sign	=	=	
greater-than sign	>	>	<
question mark	?	?	
at sign	@	@	>
uppercase letters A-Z		A-Z	
left square bracket	[	[	
backslash	\	\	
right square bracket	]	]	

caret	^	^	
horizontal bar (underscore)	_	_	
acute accent	`	`	
lowercase letters a-z		a-z	
left curly brace	{	{	
vertical bar	|			
right curly brace	}	}	
tilde	~	~	
en dash	–	–	
em dash	—	—	
nonbreaking space			
inverted exclamation		¡	
cent sign	¢	¢	
pound sterling	£	£	
general currency sign	¤	¤	
yen sign	¥	¥	
broken vertical bar	¦	¦	
section sign	§	§	
umlaut	¨	¨	
copyright	©	©	
feminine ordinal	ª	ª	
left angle quote	«	«	
not sign	¬	¬	©
soft hyphen			
registered trademark	®	®	
macron accent	¯	¯	
degree sign	°	°	
plus or minus	±	±	
superscript two	²	²	
superscript three	³	³	
acute accent	´	´	
micro sign	µ	µ	
paragraph sign	¶	¶	

middle dot	·	·	
cedilla	¸	¸	
superscript one	¹	¹	
masculine ordinal	º	º	
right angle quote	»	»	
one-fourth	¼	¼	
one-half	½	½	
three-fourths	¾	¾	
inverted question mark	¿	¿	

נספח ד'
משאבים

במהלך הספר נעשה שימוש במספר טכנולוגיות וסביבות פיתוח כמו Visual Basic, ASP ואחרות. רשימה זו של משאבים תפנה אותך לספרים רלוונטים בעברית ובאנגלית. חלק מהספרים שצוינו באנגלית יופיעו בקרוב בעברית. הינך מוזמן להתעדכן באתר ההוצאה בכתובת http://www.hod-ami.co.il .

Visual Basic

Visual Basic 6 סדנת לימוד, הוד-עמי

Visual Basic 6 ערכת כלים, יוסי שריקי, הוד-עמי

ASP

ASP 3 ובניית אתרים ב- Visual InterDev סדנת לימוד, רונן אלמוג, הוד-עמי

ASP 3 למפתחי אתרים באינטרנט, ירון לייפנברג, אייל לייפנברג, הוד-עמי

ASP Unleashed, Stephen Walther, SAMS, ISBN: 0-672-31613-7
(ספר זה יופיע בקרוב בעברית)

JavaScript

JavaScript למפתחי אתרים באינטרנט, ירון לייפנברג, הוד-עמי

PURE JavaScript, R. Allen Wyke, SAMS, ISBN: 0-672-31547-5
(ספר זה יופיע בקרוב בעברית)

ADO

Visual Basic 6 סדנת לימוד, הוד-עמי

Access 2000 VBA המדריך השלם, הוד-עמי

ADO 2.1 Programmer's Reference, Dave Sussman, WROX, ISBN: 1-861002-68-8

MTS, COM, COM+, Enterprise Applications

Visual C++ 6 המדריך השלם, קורגלינסקי, הוד-עמי

Enterprise Application Architecture, Joseph Moniz, WROX, ISBN: 1-861002-58-0

MTS MSMQ with VB and ASP, WROX, Alex Homer, ISBN: 1-861001-46-0

ActiveX and ATL

Visual C++ 6 סדנת לימוד, הוד-עמי

Visual Basic 6 סדנת לימוד, הוד-עמי

Professional ATL COM Programming, WROX, Richard Grimes, ISBN: 1-861001-40-1

Web Design and User Interface

GUI עיצוב ממשק משתמש בסביבת Windows, הוד-עמי

Designing Web Usability, Jakob Nielsen, New Riders, ISBN: 1-56205-810-X
(ספר זה יופיע בקרוב בעברית)

XML

XML למפתחי אתרים באינטרנט (ספר זה), הוד-עמי

Professional Style Sheets for HTML & XML, WROX, ISBN: 1-861001-65-7

נספח ה'
התקליטור המצורף

הוצאת הוד-עמי מגישה לך תקליטור הכולל:

- **Microsoft SQL Server 7.0 Evaluation Software 120-Day Limit on Use**
- **קטלוג HTML** - קטלוג ספרי המחשבים האינטראקטיבי של הוצאת הוד-עמי.
הקטלוג מאפשר קריאת פרקים לדוגמה, תוכן עניינים, מגה-אינדקס ועוד.
לשם קריאת הפרקים לדוגמה יש להתקין את תוכנת Adobe Acrobat Reader אשר מצורפת בתקליטור. הוראות התקנה בהמשך.
הקטלוג מומלץ לצפייה באמצעות Internet Explorer גירסה 5.
- **קוד מקור** - קבצי קוד מקור של כל הדוגמאות שבספר.



אם מנהל התקן כונן התקליטורים המותקן הוא 16 סיביות - ייתכן שתראה רק 8 תווים ראשונים של שם הקובץ (במקרה ובמקור הוא ארוך יותר). הסיבה: מחשבים שעברו שדרוג מסביבת DOS ו/או Windows 3.1x ובהם מותקן כונן תקליטורים במהירות x4, עשויים לפעול עם מנהל התקן 16 סיביות שאינו מסוגל לזהות קבצים עם שמות ארוכים. הפתרון: להתקין מנהל התקן 32 סיביות (אם קיים) או לקנות כונן תקליטורים חדש ולוודא שמצורף אליו מנהל התקן 32 סיביות.

Microsoft SQL Server 7

הוצאת **הוד-עמי** מפיצה גירסה זו של שרת SQL כ**בונוס** ללקוחותיה, ואינה מתיימרת לגבות עבורה תשלום, ו/או לתמוך בה בשום צורה ואופן.

הגירסה המצורפת היא גרסת **Evaluation** אותה ניתן להפעיל במלואה **למשך 120 יום** מיום ההתקנה.

דרישות מערכת בהמשך.

תוכנית ההתקנה מופעלת באופן אוטומטי בעת טעינת התקליטור למחשב.

יש לקרוא בעיון את הוראות ההתקנה בספר ובקבצים המצורפים בתקליטור.

לפרטים אודות ההתקנה קרא את הקובץ readme.txt שבתיקיה Install בתקליטור

אזהרה: השימוש בתוכן תקליטור זה הוא על אחריותו הבלעדית של המשתמש. המוצרים המותקנים בתקליטור זה מסופקים באחריות החברות המייצרות אותם, בהתאם לתנאי האחריות המצורפים לכל אחד מהמוצרים. הוצאת **הוד-עמי** אינה אחראית, בכל צורה שהיא, לאופן ולטיב התוכנות המותקנות.

שים לב:

ההתקנה דורשת שהדפדפן Internet Explorer גירסה 4.01 SP1 יהיה מותקן במערכת.

אם במחשב שלך לא מותקן הדפדפן כלל, כאשר תתבקש להתקין את הדפדפן - אשר.

אם במחשב שלך מותקן דפדפן IE בגירסה ישנה יותר מ- 4 - בחר בהתקנת הדפדפן.

אם במחשב שלך מותקן דפדפן IE בגירסה מתקדמת יותר מ- 4 - דלג על שלב התקנת הדפדפן ועבור להתקנת השרת עצמו.

דרישות המערכת

יש צורך ברישיון צד לקוח (Client Access Licence).

להתקנת השרת

- מחשב עם מעבד Pentium במהירות 166MHz לפחות
- מערכת הפעלה Windows NT Server גירסה 4.0 בה מותקן Service Pack גירסה 4
- Microsoft Internet Explorer גירסה 4.01 עם Service Pack גירסה 1 (מצורף). ראה הערה קודם.
- 32MB זיכרון RAM (מומלץ לפחות 256MB) *
- נפח כונן דיסק קשיח הנדרש:
  - להתקנת השרת: 65-180MB; בערך 170MB לצורך התקנה רגילה (Typical)
  - לשירותי OLAP: 35-50MB; בערך 50MB לצורך התקנה רגילה (Typical)
  - לשאילתות באנגלית: 24-36MB; בערך 36MB לצורך התקנה רגילה (Typical)
- כונן תקליטורים
- כרטיס מסך תומך ברזולוציית VGA; מומלץ כרטיס תומך Super VGA לפחות
- עכבר Microsoft או תואם לו

הערה SQL Server 7.0 מסוגל לנצל עד ארבעה מעבדים. תמיכה במספר גדול יותר של מעבדים ניתן להשיג בגרסת Enterprise של המוצר.

לתחנת עבודה

(דרישות זהות לדרישות השרת, עם היוצאים מהכלל הבאים):

- מערכת הפעלה Windows 95, Windows 98, Windows NT Workstation 4.0, Windows NT Server 4.0 או Windows NT Server Enterprise Edition 4.0 (לכל גרסאות Windows NT נדרשת התקנת Service Pack 4 לפחות. ניתן להוריד מאתר האינטרנט של חברת Microsoft בכתובת http://www.microsoft.com/windows).
- להתקנת השרת: 65-180MB; בערך 170MB לצורך התקנה רגילה (Typical)

הערה תחנת עבודה של SQL Server 7.0 מסוגלת לנצל עד שני מעבדים.

תמיכה ברשת: תמיכה מובנית במערכות ההפעלה Windows 95, Windows 98 או Windows NT (לא נדרשת תוכנת רשת נוספת, אלא אם נעשה שימוש ב- Banyan VINES או AppleTalk ADSP; **לקוחות נתמכים:** Windows 95, Windows 98 או Windows NT Workstation, UNIX**, Apple Macintosh** ו- OS/2**.

* דרישות החומרה עשויות להיות שונות בהתאם לתצורת המחשב שלך והאפשרויות שבחרת להתקין. לרשימה מלאה ומפורטת של דרישות השרת ותחנת העבודה המומלצות עבור מספר יישומים מבוססי-אינטרנט, פנה לאתר האינטרנט של Microsoft SQL Server, בכתובת http://www.microsoft.com/sql.

** תמיכה בלקוחות אלה דורשת התקנת תוכנת לקוח ODBC של צד שלישי.

קוד התקליטור לצורך ההתקנה:
040-0285715

לפרטים אודות ההתקנה קרא את הקובץ readme.txt שבתיקיה Install בתקליטור

התחלת תהליך התקנה מתבצע באופן אוטומטי בעת הכנסת התקליטור לכונן.

לביצוע ההתקנה באופן ידני, הפעל את קובץ autorun.exe שבתיקיית השורש של התקליטור המצורף.

התקנה - Acrobat Reader

יש להתקין תוכנה זו כדי לקרוא ולהדפיס את הפרקים לדוגמה, אליהם ניתן לגשת באמצעות **קטלוג HTML** (שהתקנתו תוסבר בהמשך). התוכנה גם מאפשרת חיפוש בעברית ובאנגלית במסמך המוצג. בנוסף, בעזרת תוכנה זו תוכל לקרוא את המסמכים שהוצאה מפרסמת באתר האינטרנט. התוכנה פועלת במערכות הפעלה **Windows 95 ומעלה**.

1. לחץ על לחצן **התחל** ובחר באפשרות **הפעלה**.
2. בתיבת הטקסט הקלד את הפקודה **X:\Software\Adobe Acrobat\ARME4Heb.exe** (החלף את האות X באות המייצגת את כונן התקליטורים שלך) ולחץ על **אישור**.
3. אשף ההתקנה מתקין את הרכיבים הנדרשים. עליך ללחוץ על **Next**, **Accept** ו-**Next** פעם נוספת כדי לבצע את ההתקנה.
4. בסיום ההתקנה עשויה להופיע על המסך תיבת דו-שיח **התנגשות בין גירסאות** ומייד אחר כך להיעלם. במקומה תופיע על המסך תיבת הודעה של תוכנית ההתקנה. לחץ על **אישור** ובתיבת הדו-שיח **התנגשות בין גירסאות** ששבה להופיע לחץ על **כן**, כדי לשמור את גרסת הקובץ שלך.

קוד המקור בתקליטור המצורף

תחת התיקיה Books תוכל למצוא את התיקיה הרלוונטית לספר אליו מצורף התקליטור.

קבצי קוד המקור נמצאים בתיקיה Books\59296

קוד המקור הינו אוסף התוכניות המובאות בספר. אנו מציעים לך ללמוד אותן, וחשוב לא פחות - להריץ אותן במחשב. כדי למנוע ממך את הטרחה, הטעויות ובזבוז הזמן הכרוכים בהבאתן למצב ריצה, התוכניות מצורפות בתקליטור.

התוכניות הורצו ונבדקו, אך ייתכן שתמצא טעות כלשהי בספר או בתוכניות שבתקליטור. אנו מבקשים ממך לבדוק את הנושא בעצמך, כחלק מתהליך הלימוד שלך. נודה לך אם תודיע לנו על השגיאה, אם ישנה, כדי שנוכל לתקן בהדפסת המהדורה הבאה.

העתקת קבצי קוד המקור לדיסק הקשיח

קבצי קוד המקור נמצאים בתיקיה נפרדת תחת **התיקיה Books\59296** אשר בתקליטור.

כדי להעתיק את תכולת התיקיה הרלוונטית לספר זה:

1. לחץ על לחצן **התחל**, **תוכניות**, **סייר Windows**.
2. לחץ על כונן התקליטורים וסמן את התיקיה הרלוונטית שבתיקיה Books.
3. גרור אותה לכונן הדיסק הקשיח.

מכיון שמקור הקבצים הוא התקליטור, הם מסומנים לקריאה בלבד. יש לשנות מאפיין זה כך:

1. ב**סייר Windows** היכנס לתיקיה בדיסק הקשיח בה נמצאים הקבצים שהעתקת.
2. סמן את כולם על ידי **Ctrl+A**.
3. הצב את סמן העכבר מעל האזור המסומן ולחץ לחיצה ימנית בעכבר.
4. מהתפריט המקוצר בחר **מאפיינים**.
5. בטל את הסימון בתיבה **קריאה בלבד** (דאג שהיא תהיה ריקה).
6. לחץ על **החל**, לחץ **אישור**.

הערה: יש לסמן את הקבצים בכל תיקיה בנפרד.
בדוק האם השתנה המאפיין על ידי סימון הקובץ: לחיצה ימנית על הקובץ ומתפריט הקיצור בחירה באפשרות **מאפיינים**. שים לב שתיבת הסימון **קריאה בלבד** תהיה ריקה.

קטלוג HTML

הוצאת **הוד-עמי** גאה לבשר על **קטלוג HTML** העושה שימוש בטכנולוגיות אינטרנט מתקדמות כדי להביא לך את המידע על ספרי המחשבים המקצועיים שלנו בלחיצת לחצן העכבר.

להפעלת הקטלוג **חייב** התקליטור להימצא בכונן.

הקטלוג מומלץ לצפייה באמצעות Microsoft Internet Explorer.

בעזרת **קטלוג HTML** תוכל:

1. לעיין במידע אודות ספרי ההוצאה מתי שתרצה (לחיצה כפולה.... וזהו!).
2. לעבור במהירות ובקלות בין **הקטלוג** והיישום בו אתה עובד.
3. לעיין במידע אודות כל ספר וספר.
4. לצפות ואף להדפיס פרק לדוגמה.
5. לצפות ואף להדפיס מגה-אינדקס של הספר.
6. לגשת במהירות, בגישה אינטואיטיבית, תוך התמקדות מהירה בספר המבוקש.
7. לעיין בקטלוג בקצב אישי שלך.
8. לנווט את דרכך בקטלוג ולחזור ולהתרענן בכל נושא בכל רגע.

הקטלוג מומלץ לצפייה באמצעות Internet Explorer גירסה 5 ומעלה

הפעלת הקטלוג

1. הכנס את התקליטור לכונן.
2. פתח את **סייר Windows**.
3. בחר בכונן התקליטורים ונווט דרכך לתיקיה **HTML Catalog**.
4. לחץ לחיצה כפולה על הקובץ **index.htm**.

אינדקס

לפניך אינדקס לספר, כולו באנגלית.

שים לב, מכיון שהאינדקס באנגלית, לשם נוחיות הקריאה הוא מתחיל משמאל לימין, החל מעמוד 1 שנמצא לקראת סוף הספר ומתקדם פנימה אל תוך הספר עד עמוד 23.

אינדקס

Symbols

A

B

C

D

E

F

I

J

N

O

P

S

T

U

V

W

X-Z

הדרכה וטיפים לבניית אתר באינטרנט

נכתב על ידי **זהר עמיהוד**, מחבר הספר **HTML 4 למפתחי אתרים באינטרנט**.

רשת האינטרנט מאפשרת לך לפרסם את התכנים שלך ולהביא לידי ביטוי את היצירתיות שלך במחיר "מצחיק", אם בכלל, וגם לפנות לקהל פוטנציאלי הגדול פי כמה מכל מדיה פרסומית אחרת שהכרת עד כה.

בניית אתר באינטרנט אינה רק לצרכי האגו האישי שלך, אלא יש בזה משום תרומה לקהילה על ידי שיתוף הידע שלך. עתה יש לך הזדמנות: ליצור משהו ולהראות אותו לקהילה, להביע דעה ולהציג אותה באופן פומבי.

בניית אתר באינטרנט היא גם עניין של אומץ. כל מה שתציג באתר שלך ייחשף, פוטנציאלית, בפני מיליוני גולשים מהארץ ומהעולם: מילדים ועד מבוגרים; מכל הדתות: יהודים, נוצרים, מוסלמים ודתות אחרות; בעלי דעות שונות בנושאי: איכות הסביבה, אכילת בשר, חינוך מיני ועוד. יש בזה עוצמה שצריך להשתמש בה בתבונה. עם האומץ, צריך גם לקחת אחריות. הנה מספר עצות שיעזרו לך להחליט בעיקר מה **לא** לפרסם באתר שלך:

⇐ לעולם, אל תשים באתר תוכן (תמונות, טקסט) שאינך מוכן להראות לאמא שלך.

שים באתר תוכן (תמונות, טקסט) שתוכל להתגאות בו בפני אמך.

⇐ לעולם, אל תשים באתר תוכן שאינך מוכן לדבר עליו במקום העבודה שלך.

שים באתר תוכן שתוכל לדבר עליו עם חבריך לעבודה.

⇐ לעולם, אל תשים באתר תוכן שעשוי להיחשב כהוצאת דיבה או השמצה.

⇐ הייה עירני לגבי זכויות היוצרים של החומר בו הינך עושה שימוש. אם יש בידך תכנים מסוימים אין זה אומר שתוכל לזלזל בזכויות יוצרים.

בנוסף, כדאי שהתכנים יהיו חוקיים ויתאימו לרוח החוק, אחרת אתה צפוי למבול של תביעות שיסבכו אותך לכל שארית חייך וגם עלולים להשאיר אותך ללא פרוטה.

מעבר לכך, האינטרנט היא סביבה פתוחה וחופשית, בה תוכל להציג כל מה שעולה בדעתך: טקסט, תמונות, סרטי וידאו, קטעי קול שיצרת במו ידך, וכל דבר שניתן להעלות במדיה מגנטית.

אתר

ה**אינטרנט** (Internet) היא רשת של רשתות מחשבים המאפשרת גישה לכל אחד. זהו למעשה איחוד של הידע האנושי. **רשת מחשבים** היא קבוצת מחשבים המחוברים בדרך המאפשרת להם ל"דבר" זה עם זה. באינטרנט, המחשבים "מדברים" **TCP/IP**. מחשב ברשת הנותן שירותי מידע הכוללים: טקסט, תמונות, סרטי וידאו, קטעי קול וכדומה ו/או שירותים כגון: דואר אלקטרוני, קבוצות דיון, אחסון אתרים וכדומה, נקרא **שרת** (Server). מחשב ברשת המקבל שירות נקרא **לקוח** (Client).

לאחר שהתחברת לאינטרנט (כמשתמש), תוכל לגשת למידע הנמצא במחשבי השרת. המידע במחשב שרת מאורגן במבנה של **אתר** (Web Site). אתר מורכב מ**דפים** (Pages) המקושרים ביניהם, כך שבלחיצת עכבר תוכל להפעיל **קישור** (Link) שבעזרתו ניתן לעבור מדף לדף. בעזרת קישור ניתן לעבור לדף אחר באותו אתר או לדף אחר באתר אחר הנמצא פיסית על אותו שרת, או לאתר אחר הנמצא בשרת אחר ברשת.

אחסון

אחד הדברים שתצטרך לחשוב עליו הוא המיקום בו יאוחסן האתר שלך. ספקי השירות לאינטרנט (ISPs) מאפשרים לך לבנות ולאחסן את האתר הפרטי שלך במחשביהם. יש חברות עסקיות רבות המוכרות שטחי אחסון, ויש חברות שנותנות בחינם (כן בחינם!) שטחי אחסון עבור האתר שלך.

להלן נקודות שיש לשקול בעת בחירת מקום אחסון:

⇐ **מחיר** - בדרך כלל התשלום עבור אחסון האתר הוא תשלום חודשי שיש בו מרכיב קבוע כלשהו ומרכיב משתנה, על פי גודל השטח המוקצה לאתר (דפי HTML, הקבצים הנלווים להם ותיבות הדואר האלקטרוני).

⇐ **חיבוריות, אמינות** - אין טעם לשים את האתר במחשב שלא ניתן לגשת אליו ו/או שהגישה אליו איטית. זכור, שהגולשים הפוטנציאלים שלך יתרגזו אם ינסו לגשת לאתר שלך ויקבלו את ההודעה הלא נעימה - Access Denied (הגישה נדחית). הם גם לא ירצו להמתין יותר מ-7 שניות להעלאת הדף, ויעברו לאתר אחר.

⇐ **שירות** - השירות מתבטא בתמיכה בבניית האתר. אם תיתקל בקושי או תרצה להיוועץ עם התמיכה, האם תוכל? יש אתרים שבשמחה יאחסנו עבורך את האתר, אבל - אתה לעצמך; ויש אתרים שיספקו לך תמיכה.

⇐ **שירותים נוספים** - שירותים הניתנים לך כבונה אתר כוללים כלים לבניית אתר, תוכנה סטטיסטית לבדיקת מספר המבקרים באתר, תוכנה לניהול תיבות דואר, כלים לבדיקת האתר (למשל, בדיקת קישוריות, כדי לוודא שכל הקישורים שכתבת בדפי HTML שלך אכן מובילים למקום כלשהו).

נפח אחסון

הנפח שתזדקק לו לאחזקת האתר תלוי בתכנים שהינך רוצה לפרסם. אם האתר שלך יכלול סרטי וידאו, תמונות סרוקות, קבצי קול, קבצי Flash ומולטימדיה, תזדקק לעשרות MB (מיליוני בתים). יש מגבלה על נפח האחסון שרכשת, או שקיבלת חינם. כאשר אתה משלם, תוכל לקבוע את נפח האחסון הדרוש לך. וכאשר אתה מקבל נפח אחסון בחינם, אתה מקבל מה שנותנים ויהיה עליך להסתדר עם מגבלה זו.

כאשר אתה מאחסן אתר בשרת, לרוב אתה מקבל גם שירות דואר אלקטרוני. נפח האחסון עבורך כולל את הקבצים לאתר ולדואר אלקטרוני. אין בזה רע, רק צריך להיות מודע לכך.

תחום - Domain

שם אתר, כמו למשל www.hod-ami.co.il, הוא אחד המרכיבים החשובים והשווים כסף. שם קצר וקליט לאתר שלך הוא יתרון שמצריך מחשבה ועולה כסף.

- **שם קצר** - למד מהניסיון של חנות הספרים www.baranesandnobel.com, שהרבה יותר קל לכתוב את שמה כך: www.bn.com.
- **שם בעל משמעות** - לא חובה, אבל רצוי. אם האתר שלך הוא בנושא מזון, כדאי שיהיה לו שם כמו food, efood, food4u, netfood וכדומה, כמובן עם הפתיחה www והסיומת co.il או com.

את שם התחום (domain) צריך לרכוש ולשלם עבורו. זה לא תשלום גבוה, אבל כדאי למהר, לפני שמישהו אחר ירכוש. שם התחום נרכש לתקופה של שנתיים וצריך לחדשו מעת לעת.

אני יודע מה אתה חושב - אתה חושב איך לעשות מזה כסף והרבה. "ארכוש שמות תחום כהשקעה, וכשמישהו ירצה את אותו שם תחום, אמכור לו ברווח נאה". ובכן... זאת אפשרות, אבל כדאי לקרוא את כל הפרטים הקטנים בעת רכישת השם: מה מותר ומה אסור, מהן הזכויות ומהן החובות.

שם תחום ניתן לך באתרים בהם תקבל מקום אחסון חינם. ולך יש שליטה מעטה, אם בכלל, על השם. לעומת זאת, באתרים בהם תרכוש מקום אחסון בכסף מלא, תוכל גם לרכוש שם תחום משלך.

תכנון האתר

התכנים שתשים באתר חשובים, אך לא פחות מזה - חשובה העטיפה - כיצד נראה האתר. החופש ליצור אין פירושו שתוכל להקל ראש בבדיקת הנכונות התחבירית של החומר באתר, או להציג בו תמונות שיקשה להבין מה בדיוק צולם בהן. מכיון שאתה נמצא בסביבת מולטימדיה, חייב האתר שלך להראות יכולת שכזו, ולו באופן חלקי.

עיצוב ותוכן

מה לשים באתר שלי? באינטרנט, וזה כבר ידוע לך, יש **הכל!** יש אתרים מלאי תוכן ויש כאלה שאינך מבין על מה ולמה, יש אתרים מעוצבים ויש כאלה רק עם טקסט שחור ללא טיפת צבע, יש אתרים עם "פירוטכניקה": מצגות Flash, אנימציות, מוסיקה ועוד. בקיצור, יש אתרים עם הרבה תנועה ויש אתרים סטטיים. אתה בונה את האתר הפרטי שלך, וזהו בוודאי האתר הראשון שלך, ולכן ראוי לשים לב לכל פרט.

אתר מציצני לחייך, המכיל תמונות שלך (אוכל, שותה, עובד בגינה, משחק כדורגל) יכול להוות לדעתך מקור משיכה למיליוני גולשים בעולם, אבל ספק רב אם הם יישארו שם יותר מ-2 שניות.

בוודאי יש לך תחביב כלשהו או כישרון שיכול לעניין אנשים אחרים. בוודאי יש לך משהו מעניין לומר וגם דרך מקורית להציג את טיעונך, למרות שהנושא אינו חדש. האינטרנט הוא המקום להציג את מה שיש לך לומר, ובגדול.

מספר טיפים כלליים:

⇐ זכור, שאת התמונות שתפרסם באתר יוכלו אחרים להוריד. אם אתה רוצה להימנע מבעיות, ודא שהתמונות מתאימות לכל אחד.

⇐ בנה אתר שניתן יהיה לצפות בו בעזרת Internet Explorer וגם בעזרת Netscape. בכל מקרה ציין בעמוד הפתיחה מהו הדפדפן שהינך ממליץ עליו לצפייה.

⇐ ציין בעמוד הפתיחה את רזולוציית המסך הנדרשת לצפייה באתר. אם אתה בונה אתר ברזולוציה של 800x600 והרזולוציה במחשב של הגולש לאתר שלך היא רק 640x480, הוא לא יראה את כל מה שהתכוונת להראות לו.

⇐ כשאתה בונה את האתר, חשוב על "הגולש הסביר" באתר שלך בכל הנוגע לאמצעי הניווט. אל תיתן לו ללכת לאיבוד בין הדפים השונים.

⇐ קשר את כל הדפים באתר, כך שהגולש באתר לא יהיה חייב להשתמש בלחצני Back ו-Forward של תוכנת הדפדפן.

⇐ היצירתיות חשובה, אבל צריך גם לשמור על טוב טעם. זה שיש לך אפשרות להשתמש בצבעים (וליתר דיוק ב-**216** גוונים), אין זה אומר שאתה יכול ל"שפוך" צבע כאוות נפשך.

אתה רוצה שיגיעו אליך גולשים ויישארו לשוטט באתר שלך. הרצון שלהם להישאר שם תלוי מאוד בתוכן וכיצד הוא מוצג, ובעניין הזה עליך להיות שונה. באינטרנט יש מיליוני אתרים של אנשים פרטיים שבנו אותם בצורה שבלונית ומשעממת ואני בטוח שאינך רוצה להצטרף אליהם.

חשוב **איך ל"עצור" את הגולש** באינטרנט שמגיע לאתר שלך. אם האתר שלך (וכרגע לא חשוב תוכנו) ייראה, מבחינה עיצובית, כמו אלפי אתרים אחרים - הוא לא יעצור אצלך. לדוגמה, תמונה של פני אשה המתגלחת בסכין גילוח תסב את תשומת הלב. השילוב של

פנים, אשה וסכין גילוח הינו מסקרן ויש סיכוי שהוא יעצור, ולו לרגע, את הגולש הצופה בו. חשוב על זה כשאתה בונה את האתר שלך, ובמיוחד בעמוד הראשון שנגלה לעיני הגולש.

לעיצוב יש "מחיר". לאתר עמוס במולטימדיה, יישומוני Java (הנקראים Applets), סרטוני Flash ושאר "פיצוצים", יש זמן טעינה רב יותר - והדבר עלול להבריח את הגולשים אל אתרים אחרים. בעידן הפיצה, בו אנו רוצים לקבל הכל ומייד, צריך מנהל האתר העסקי לחשוב איך מעבירים את הסחורה כמה שיותר מהר אל הלקוח.

אל תבטיח מה שאינך יכול לקיים. בניסיון למשוך את הגולשים אל תתאר את האתר שלך כ"מקום האולטימטיבי, המלא והשלם לטיפול בבעלי חיים" אם אתה רק מתכוון לשים תמונה של הכלב שלך והסבר קצר איך אתה מטפל בו.

דאג לעדכון האתר. זו חוכמה קטנה מאוד לבנות אתר ולא לתחזק אותו מעת לעת. קח לדוגמה את החנויות בקניון, שבניסיון שלהם למשוך לקוחות חדשים ולהכניס שוב את הלקוחות הקיימים, הן משנות מדי פעם את התצוגה בחלון הראווה ואת הסידור הפנימי בחנות, כדי להציג רעננות וחדשנות.

דאג לתאריך מעודכן. פרט לשינוי המראה והתוכן, דאג לציין את תאריך העדכון האחרון, ובדרך כלל בדף הראשי. בדפים שאין משמעות לתאריך, אל תוסיף אותו. מה לעשות, ככלל, אנו מעריכים יותר חומר חדש על פני חומר משנים עברו.

תן לגולשים אפשרות להתקשר איתך, ובוודאי תן להם אפשרות לשלוח לך דואר אלקטרוני. אם תרצה לשפר את הדו-שיח ביניכם, תן להם למלא טופס בו יוכלו להתבטא. והדבר החשוב, קרא לעיתים תכופות את הדואר הנשלח אליך וגם רצוי שתשיב עליו, או על רובו.

אם הגולש זקוק לתוכנות אחרות לצורך השימוש בתכנים שבאתר, ציין זאת באופן ברור והצב קישורים לאתרים מהם ניתן להוריד את התוכנות. זה נכון לתוכנות כמו RealAudio, Adobe Acrobat Reader, Microsoft Word Viewer ואחרות.

כלים זמינים ופשוטים לבניית אתר באינטרנט - I

באינטרנט תוכל למצוא תוכנות רבות ושונות לבניית אתרים: Freeware, Shareware, Trial Version ועוד.

דפדפן (Browser)

רצוי שיהיו בידך שני הדפדפנים העיקריים: Internet Explorer ו-Netscape כדי שתוכל לבחון כיצד ייראה האתר שלך בעיני הגולשים.

עורך HTML

דפים באינטרנט (Web Pages) נבנים בעזרת שפת תגיות הנקראת **HTML** (זו אינה שפת תכנות). בעזרת HTML תקבע כיצד ייראה תוכן הדף: טקסט (גודל גופן, הדגשה וכו'), תמונות (גודל, מסגרת, רווח בין תמונה לטקסט וכו') ושאר מרכיבים.

כלי פיתוח שונים מאפשרים לבנות דפי HTML ממש בקצות אצבעותיך, אבל, רובם לא יודעים עברית! שים לב! כדי להשתמש בהם צריך ללמוד תוכנה חדשה ולוודא שמה שיצרת ניתן לצפייה בדפדפנים השונים. אחת הדרכים הפשוטות הזמינות, אם ברשותך Word 97 או Word 2000, היא ליצור מסמך במעבד תמלילים ולשמור אותו כדף HTML - מגניב! אהה?!? ובכן, לקלות ולמהירות הזאת יש "מחיר", והוא:

⇐ בתהליך ההסבה נוצר קובץ HTML גדול, שמגדיל באופן משמעותי את זמן טעינת הדף,

⇐ רק הגולשים עם תוכנת Internet Explorer יוכלו לצפות בדף (הכתוב עברית) ו"לא יעזור בית דין" - הגולשים עם Netscape יראו ℜϒ∉⊇Ξϑ Ω.

עורך טקסט

תוכנת Notepad (פנקס הרשימות) נמצאת בכל מחשב עם מערכת הפעלה Windows בכל הגרסאות, כך שהיא זמינה לכולם. כדי לכתוב עם עורך טקסט, יש ללמוד את שפת HTML ואת זה ניתן לעשות עם הספר **HTML 4 למפתחי אתרים באינטרנט**.

תוכנה גרפית

האתר שלך יכלול בוודאי מספר תמונות, אותן יש לסרוק ולעבד. עיבוד התמונה כולל ביצוע תיקונים בתמונה (למשל, מחיקת כתם או צביעה מחדש של עיניים אדומות). בנוסף, יהיה צורך ביצירת תמונות או לחצנים או באנרים. לכל אלה מומלץ להשתמש בתוכנה Paint Shop Pro הנמצאת בגרסת ניסיון בתקליטור המצורף לספר **Paint Shop Pro 6 עיצוב גרפיקה באינטרנט.**

עברית

כדי שמרבית הגולשים באינטרנט יוכלו לצפות באתר שלך הכתוב עברית, יהיה עליך להשתמש בתוכנות נוספות. על תוכנות אלו ועל השימוש בהן תוכל לקרוא בספר **HTML 4 למפתחי אתרים באינטרנט.**

תוכנת FTP

את הקבצים המרכיבים את האתר שבנית, ושנמצאים במחשב האישי שלך, יש להעביר למחשב בו יאוחסן האתר. תוכנת FTP מאפשרת גישה למחשב באינטרנט באופן שמאוד מזכיר את סייר Windows הנמצא כחלק ממערכת ההפעלה Windows בכל גירסה.

אחת התוכנות היא WS_FTP הנמצאת בתקליטור המצורף לספר **HTML 4 למפתחי אתרים באינטרנט**.

לחלק מאתרי האחסון יש מנגנון להעברת קבצים משלהם. פעל לפי הנחיות האתר.

פרסום האתר

אחת המטרות בבניית אתר היא שיגיעו אליו הגולשים. פרסום האתר על ידי באנרים עולה הרבה כסף, אבל יש אפשרות לפרסום חינם (להלן רשימה חלקית):

⇐ רישום במנועי חיפוש ישראליים (ולא רק),

⇐ השתתפות בקבוצות דיון,

⇐ החלפת באנרים עם אתרים בעלי עניין דומה.

על אמצעי הפרסום של האתר שלך קרא בספר **HTML 4 למפתחי אתרים באינטרנט**.

כלי תכנות לבניית אתר באינטרנט - II

JavaScript - שפת תכנות הנכתבת במסגרת דף HTML שבעזרתה ניתן לנהל דו-שיח עם המשתמש בצורה הרבה יותר טובה מאשר עם קישורים בלבד. שילוב שפת תכנות זו מאפשר יצירת אינטראקטיביות בין האתר למשתמש בו. להוספת JavaScript אינך צריך כל תוכנה אחרת, עורך הטקסט שברשותך יספיק.

הנה כמה מהדברים שניתן לעשות עם JavaScript :

⇐ **בניית טופס**. כאשר המשתמש ילחץ על לחצן **שלח טופס** תוכל לבדוק מה רשם בו המשתמש ולהעיר לו בהתאם, אם לא מילא שדה בטופס או מילא ערך שגוי. תוכל גם להציג בפניו שוב את הטופס כדי שיתקן את הטעון תיקון.

⇐ **מקום הימצאות הסמן**. תוכל לדעת היכן נמצא הסמן ובהתאם לכך להציג תכנים מבלי שהמשתמש לחץ בעכבר. מספיק שהוא העביר את הסמן מעל מקום שקבעת. למשל, כאשר המשתמש מעביר את הסמן מעל טקסט מסוים, רקע הטקסט משתנה. למשל, כאשר המשתמש מעביר את הסמן מעל תמונה מסוימת, היא מתחלפת בתמונה אחרת.

⇐ **עבודה עם חלונות**. הפעלת קישור בדף HTML מציגה תוכן של דף HTML באותו חלון או בחלון חדש. עם JavaScript תוכל לשלוט על גודל החלון, כמו גם על הלחצנים ושורת הכותרת שלו.

היכרות עם JavaScript תוכל לעשות בעזרת הספר **HTML 4 למפתחי אתרים באינטרנט**.

לימוד JavaScript תוכל לעשות בעזרת הספר **JavaScript למפתחי אתרים באינטרנט**.

Java - שפת תכנות המיועדת לבניית יישומונים (Applets), במיוחד עבור אתרי אינטרנט. לצורך כתיבת יישומון Java יש צורך בערכת פיתוח תוכנה שאינה כלולה בכלים שהוזכרו עד כה, אותה ניתן להוריד בחינם מהאינטרנט. שפת Java הינה שפת תכנות, כמו למשל ++C. תוכל בעזרתה לכתוב כל תוכנית שתעלה בדעתך, החל מתוכנית המקבלת מספר מהמשתמש וכופלת אותו ב-2, דרך בניית טפסים לקליטת נתונים, משחק מחשב, ועד תוכנית לניהול חשבון בנק. אם אתה מעוניין, מסיבה כלשהי, לכתוב תוכנית לאינטרנט שתוכל לפעול מבלי שלמשתמש תהיה יכולת להסתכל בקוד - השתמש ב- Java.

לימוד Java תוכל לעשות בעזרת הספר **Java 2 למפתחי אתרים באינטרנט**.

DHTML - המשך המגמה של יצירת דפי HTML אינטראקטיביים ודינמיים ככל האפשר. שימוש ברכיבי מערכת מובנים (חלק ממערכת Windows) ורתימה שלהם לדפי HTML. נחמד ויפה, אבל מכיון שזוהי טכנולוגיה של Microsoft, משתמשי Netscape לא יראו דבר, כלום!

היכרות עם DHTML תוכל לעשות בעזרת הספר **HTML 4 למפתחי אתרים באינטרנט**.

VBScript - שפת תסריט, דומה בגישתה ל-JavaScript אבל היא של Microsoft. המשמעות היא שמי שמשתמש בדפדפן Netscape לא יוכל ליהנות מטכנולוגיה זו. לכן, אם אתה מתכוון להשתמש בשפת תסריט במסגרת דפי HTML שהינך כותב, עשה זאת עם JavaScript.

היכרות עם VBScript תוכל לעשות בעזרת הספר **HTML 4 למפתחי אתרים באינטרנט**.

ASP - עוד טכנולוגיה מבית Microsoft שעובדת רק על שרתי Web מסוג IIS (תוכנת שרת מבית Microsoft) המאפשרת, בעיקר, ניהול מסדי נתונים באינטרנט, החל ממועדון לקוחות, דרך חנות ועד ניהול חשבונות בנק. ASP זה המנוע שמאחורי הטופס והלחצן שמבצע פעולות ברקע, בעיקר של טיפול בנתונים : הוספה, מחיקה, עדכון, יצירת דוח או ביצוע שאילתה.

לימוד ASP תוכל לעשות בעזרת הספר **ASP 3 למפתחי אתרים באינטרנט** (לספר מצורפת תוכנת PWS להרצת ASP. הספר כולל גם לימוד VBScript במידה המספיקה לכתיבת ASP).

Flash - בניסיון להעלות עוד את הרף בכל הקשור למולטימדיה באינטרנט, באה לעולם תוכנת Flash. בעזרתה יוצרים הנפשה, הכוללת פעולות אינטראקטיביות וטפסים המיועדים לאינטרנט, והכל עשיר בתנועה ובצליל ונטען במהירות. סרטי Flash בונים בעזרת תוכנת Flash אותה יש לרכוש.

לימוד Flash תוכל לעשות בעזרת הספר **Flash 4 למפתחי אתרים באינטרנט** (בתקליטור המצורף לספר נמצאת תוכנת Flash בגרסת ניסיון).

חינם

המילה ״חינם״ היא מילת מפתח באינטרנט, אבל... יש לשים לב. דע לך, אם לא ידעת את זה עד כה, שחינם זה לא חינם! האם חשבת למה לאפשר לך שירות חינם?

אחסון חינם - האחסון חינם וזה נכון וגם סופי, אבל... הגולשים שלך יראו פרסומות שנשתלו בדפי HTML שלך ובמהלך הגלישה ייפתחו חלונות עם תוכן פרסומי.

לחצנים בחינם - ״חינם״ פה יכול להתבטא שאכן תוכל לעשות שימוש בגרפיקה של הלחצן, אבל... יהיה עליך לציין במפורש את מקורם בצירוף קישור לאתר עם תמונה גרפית קטנה (באנר). בסופו של דבר האתר שלך יכול להיראות כמו אתר חסות.

דואר אלקטרוני חינם - הכתובת האלקטרונית שלך שווה הרבה כסף. בין הדואר שתקבל תמצא גם פרסומות של האתר, הנותן לך את תיבת הדואר בחינם, ו/או של מפרסמים אחרים המשתמשים בשירותיו ומשלמים הרבה מאוד כסף.

שירותים נוספים חינם - שירותים נוספים יכולים לכלול מועדון לקוחות, תוכנת צ׳ט, קבוצות דיון, לוחות מודעות ועוד שירותי קהילה וירטואלית, וצריך להבין בדיוק מהו התשלום עבור ה״חינם״ בכל שירות ושירות. לדוגמה, אחד השירותים הניתנים חינם הוא רישום למועדון לקוחות. התהליך עובד להפליא, אבל בכל פעם שהלקוח שלך יירשם : ימלא את הטופס וילחץ על לחצן שלח טופס (Submit), פרטי הרישום יגיעו אליך והלקוח יועבר לאתר המספק לך את אותו שירות חינם. לזה התכוונת?